Chapter 1
Introduction

Computing has become a necessary means of scientific study. Even in ancient times, the quantification of gained knowledge played an essential role in the further development of mankind. In this chapter, we will discuss the role of computation in advancing scientific knowledge and outline the current status of computational science. We will only provide a quick tour of the subject here. A more detailed discussion on the development of computational science and computers can be found in Moreau (1984) and Nash (1990). Progress in parallel computing and global computing is elucidated in Koniges (2000), Foster and Kesselman (2003), and Abbas (2004).

1.1 Computation and science

Modern societies are not the only ones to rely on computation. Ancient societies also had to deal with quantifying their knowledge and events. It is interesting to see how the ancient societies developed their knowledge of numbers and calculations with different means and tools. There is evidence that carved bones and marked rocks were among the early tools used for recording numbers and values and for performing simple estimates more than 20 000 years ago.

The most commonly used number system today is the *decimal system*, which was in existence in India at least 1500 years ago. It has a radix (base) of 10. A number is represented by a string of figures, with each from the ten available figures (0–9) occupying a different decimal level. The way a number is represented in the decimal system is not unique. All other number systems have similar structures, even though their radices are quite different, for example, the *binary system* used on all digital computers has a radix of 2. During almost the same era in which the Indians were using the decimal system, another number system using dots (each worth one) and bars (each worth five) on a base of 20 was invented by the Mayans. A symbol that looks like a closed eye was used for zero. It is still under debate whether the Mayans used a base of 18 instead of 20 after the first level of the hierarchy in their number formation. They applied these dots and bars to record multiplication tables. With the availability of those tables, the

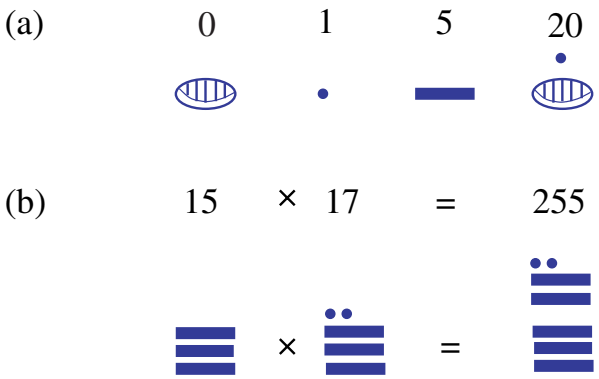

Fig. 1.1 The Mayan number system: (a) examples of using dots and bars to represent numbers; (b) an example of recording multiplication.

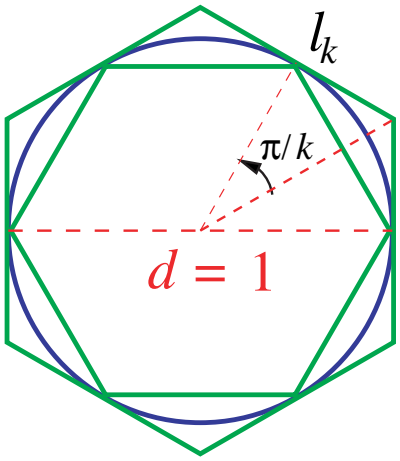

Fig. 1.2 A circle inscribed and circumscribed by two hexagons. The inside polygon sets the lower bound while the outside polygon sets the upper bound of the circumference.

Mayans studied and calculated the period of lunar eclipses to a great accuracy. An example of *Mayan number system* is shown in Fig. 1.1.

One of the most fascinating numbers ever calculated in human history is π, the ratio of the circumference to the diameter of the circle. One of the methods of evaluating π was introduced by Chinese mathematician Liu Hui, who published his result in a book in the third century. The circle was approached and bounded by two sets of regular polygons, one from outside and another from inside of the circle, as shown in Fig. 1.2. By evaluating the side lengths of two 192-sided regular polygons, Liu found that $3.1410 < \pi < 3.1427$, and later he improved his result with a 3072-sided inscribed polygon to obtain $\pi \simeq 3.1416$. Two hundred years later, Chinese mathematician and astronomer Zu Chongzhi and his son Zu Gengzhi carried this type of calculation much further by evaluating the side lengths of two 24 576-sided regular polygons. They concluded that $3.141\,592\,6 < \pi < 3.141\,592\,7$, and pointed out that a good approximation was given by

$\pi \simeq 355/113 = 3.141\,592\,9\ldots$. This is extremely impressive considering the limited mathematics and computing tools that existed then. Furthermore, no one in the next 1000 years did a better job of evaluating π than the Zus.

The Zus could have done an even better job if they had had any additional help in either mathematical knowledge or computing tools. Let us quickly demonstrate this statement by considering a set of evaluations on polygons with a much smaller number of sides. In general, if the side length of a regular k-sided polygon is denoted as l_k and the corresponding diameter is taken to be the unit of length, then the approximation of π is given by

$$\pi_k = k l_k . \tag{1.1}$$

The exact value of π is the limit of π_k as $k \to \infty$. The value of π_k obtained from the calculations of the k-sided polygon can be formally written as

$$\pi_k = \pi_\infty + \frac{c_1}{k} + \frac{c_2}{k^2} + \frac{c_3}{k^3} + \cdots , \tag{1.2}$$

where $\pi_\infty = \pi$ and c_i, for $i = 1, 2, \ldots, \infty$, are the coefficients to be determined. The expansion in Eq. (1.2) is truncated in practice in order to obtain an approximation of π. Then the task left is to solve the equation set

$$\sum_{j=1}^{n} a_{ij} x_j = b_i , \tag{1.3}$$

for $i = 1, 2, \ldots, n$, if the expansion in Eq. (1.2) is truncated at the $(n-1)$th order of $1/k$ with $a_{ij} = 1/k_i^{j-1}$, $x_1 = \pi_\infty$, $x_j = c_{j-1}$ for $j > 1$, and $b_i = \pi_{k_i}$. The approximation of π is then given by the approximate π_∞ obtained by solving the equation set. For example, if $\pi_8 = 3.061\,467$, $\pi_{16} = 3.121\,445$, $\pi_{32} = 3.136\,548$, and $\pi_{64} = 3.140\,331$ are given from the regular polygons inscribing the circle, we can truncate the expansion at the third order of $1/k$ and then solve the equation set (see Exercise 1.1) to obtain π_∞, c_1, c_2, and c_3 from the given π_k. The approximation of $\pi \simeq \pi_\infty$ is 3.141 583, which has five digits of accuracy, in comparison with the exact value $\pi = 3.141\,592\,65\ldots$. The values of π_k for $k = 8$, 16, 32, 64 and the extrapolation π_∞ are all plotted in Fig. 1.3. The evaluation can be further improved if we use more π_k or ones with higher values of k. For example, we obtain $\pi \simeq 3.141\,592\,62$ if $k = 32$, 64, 128, 256 are used. Note that we are getting the same accuracy here as the evaluation of the Zus with polygons of 24 576 sides.

In a modern society, we need to deal with a lot more computations daily. Almost every event in science or technology requires quantification of the data involved. For example, before a jet aircraft can actually be manufactured, extensive computer simulations in different flight conditions must be performed to check whether there is a design flaw. This is not only necessary economically, but may help avoid loss of lives. A related use of computers is in the reconstruction of an unexpectred flight accident. This is extremely important in preventing the same accident from happening again. A more common example is found in the cars

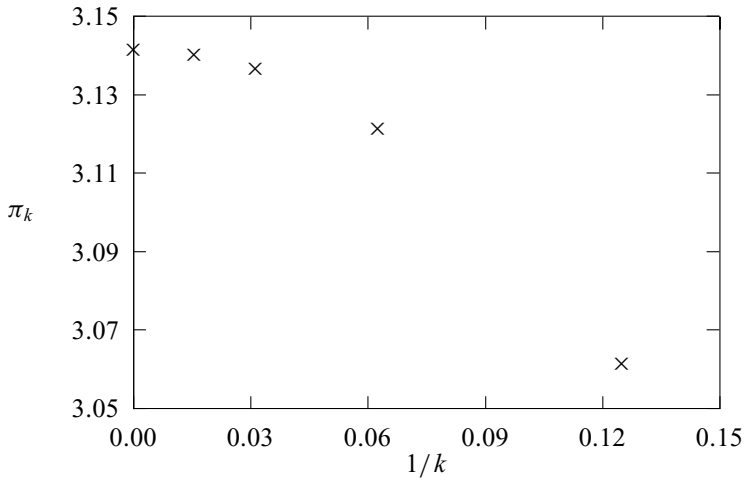

Fig. 1.3 The values of π_k, with $k = 8, 16, 32$, and 64, plotted together with the extrapolated π_∞.

that we drive, which each have a computer that takes care of the brakes, steering control, and other critical components. Almost any electronic device that we use today is probably powered by a computer, for example, a digital thermometer, a DVD (digital video disc) player, a pacemaker, a digital clock, or a microwave oven. The list can go on and on. It is fair to say that sophisticated computations delivered by computers every moment have become part of our lives, permanently.

1.2 The emergence of modern computers

The advantage of having a reliable, robust calculating device was realized a long time ago. The early *abacus*, which was used for counting, was in existence with the Babylonians 4000 years ago. The Chinese abacus, which appeared at least 3000 years ago, was perhaps the first comprehensive calculating device that was actually used in performing addition, subtraction, multiplication, and division and was employed for several thousand years. A traditional Chinese abacus is made of a rectangular wooden frame and a bar going through the upper middle of the frame horizontally. See Fig. 1.4. There are thirteen evenly spaced vertical rods, each representing one decimal level. More rods were added to later versions. On each rod, there are seven beads that can be slid up and down with five of them held below the middle bar and two above. Zero on each rod is represented by the beads below the middle bar at the very bottom and the beads above at the very top. The numbers one to four are repsented by sliding one–four beads below the middle bar up and five is given be sliding one bead above down. The numbers six to nine are represented by one bead above the middle bar slid down and one–four beads below slid up. The first and last beads on each rod are never used or are only used cosmetically during a calculation. The Japanese abacus, which was modeled on the Chinese abacus, in fact has twenty-one rods, with only five beads

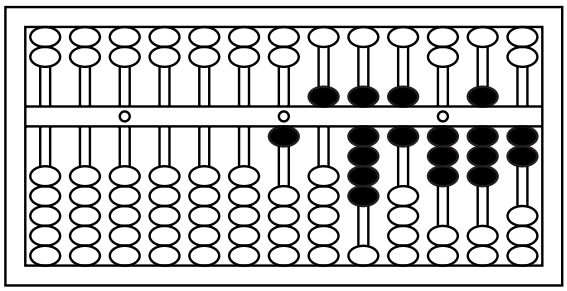

Fig. 1.4 A sketch of a Chinese abacus with the number 15 963.82 shown.

on each rod, one above and four below the middle bar. Dots are marked on the middle bar for the decimal point and for every four orders (ten thousands) of digits. The abacus had to be replaced by the slide rule or numerical tables when a calcualtion went beyond the four basic operations even though later versions of the Chinese abacus could also be used to evaluate square roots and cubic roots.

The *slide rule*, which is considered to be the next major advance in calculating devices, was introduced by the Englishmen Edmund Gunter and Reverend William Oughtred in the mid-seventeenth century based on the logarithmic table published by Scottish mathematician John Napier in a book in the early seventeenth century. Over the next several hundred years, the slide rule was improved and used worldwide to deliver the impressive computations needed, especially during the Industrial Revolution. At about the same time as the introduction of the slide rule, Frenchman Blaise Pascal invented the *mechanical calculating machine* with gears of different sizes. The mechanical calculating machine was enhanced and applied extensively in heavy-duty computing tasks before digital computers came into existence.

The concept of an all-purpose, automatic, and programmable computing machine was introduced by British mathematician and astronomer Charles Babbage in the early nineteenth century. After building part of a mechanical calculating machine that he called a *difference engine*, Babbage proposed constructing a computing machine, called an *analytical engine*, which could be programmed to perform any type of computation. Unfortunately, the technology at the time was not advanced enough to provide Babbage with the necessary machinery to realize his dream. In the late nineteenth century, Spanish engineer Leonardo Torres y Quevedo showed that it might be possible to construct the machine conceived earlier by Babbage using the electromechanical technology that had just been developed. However, he could not actually build the whole machine either, due to lack of funds. American engineer and inventor Herman Hollerith built the very first electromechanical *counting machine*, which was commisioned by the US federal government for sorting the population in the 1890 American census. Hollerith used the profit obtained from selling this machine to set up a company, the Tabulating Machine Company, the predecessor of IBM (International

Business Machines Corporation). These developments continued in the early twentieth century. In the 1930s, scientists and engineers at IBM built the first difference tabulator, while researchers at Bell Laboratories built the first relay calculator. These were among the very first electromechanical calculators built during that time.

The real beginning of the computer era came with the advent of electronic digital computers. John Vincent Atanasoff, a theoretical physicist at the Iowa State University at Ames, invented the electronic digital computer between 1937 and 1939. The history regarding Atanasoff's accomplishment is described in Mackintosh (1987), Burks and Burks (1988), and Mollenhoff (1988). Atanasoff introduced vacuum tubes (instead of the electromechanical devices used earlier by other people) as basic elements, a separated memory unit, and a scheme to keep the memory updated in his computer. With the assistance of Clifford E. Berry, a graduate assistant, Atanasoff built the very first electronic computer in 1939. Most computer history books have cited ENIAC (Electronic Numerical Integrator and Computer), built by John W. Mauchly and J. Presper Eckert with their colleagues at the Moore School of the University of Pennsylvania in 1945, as the first electronic computer. ENIAC, with a total mass of more than 30 tons, consisited of 18 000 vacuum tubes, 15 000 relays, and several hundred thousand resistors, capacitors, and inductors. It could complete about 5000 additions or 400 multiplications in one second. Some very impressive scientific computations were performed on ENIAC, including the study of nuclear fission with the liquid drop model by Metropolis and Frankel (1947). In the early 1950s, scientists at Los Alamos built another electronic digital computer, called MANIAC I (Mathematical Analyzer, Numerator, Integrator, and Computer), which was very similar to ENIAC. Many important numerical studies, including Monte Carlo simulation of classical liquids (Metropolis *et al.*, 1953), were completed on MANIAC I.

All these research-intensive activities accomplished in the 1950s showed that computation was no longer just a supporting tool for scientific research but rather an actual means of probing scientific problems and predicting new scientific phenomena. A new branch of science, *computational science*, was born. Since then, the field of scientific computing has developed and grown rapidly.

The computational power of new computers has been increasing exponentially. To be specific, the computing power of a single computer unit has doubled almost every 2 years in the last 50 years. This growth followed the observation of Gordon Moore, co-founder of Intel, that information stored on a given amount of silicon surface had doubled and would continue to do so in about every 2 years since the introduction of the silicon technology (nicknamed Moore's law). Computers with transistors replaced those with vacuum tubes in the late 1950s and early 1960s, and computers with very-large-scale integrated circuits were built in the 1970s. Microprocessors and vector processors were built in the mid-1970s to set the

stage for personal computing and supercomputing. In the 1980s, microprocessor-based personal computers and workstations appeared. Now they have penetrated all aspects of our lives, as well as all scientific disciplines, because of their affordability and low maintenance cost. With technological breakthroughs in the RISC (Reduced Instruction Set Computer) architecture, cache memory, and multiple instruction units, the capacity of each microprocessor is now larger than that of a supercomputer 10 years ago. In the last few years, these fast microprocessors have been combined to form parallel or distributed computers, which can easily deliver a computing power of a few tens of gigaflops (10^9 floating-point operations per second). New computing paradigms such as the Grid were introduced to utilize computing resources on a global scale via the Internet (Foster and Kesselman, 2003; Abbas, 2004).

Teraflop (10^{12} floating-point operations per second) computers are now emerging. For example, Q, a newly installed computer at the Los Alamos National Laboratory, has a capacity of 30 teraflops. With the availability of teraflop computers, scientists can start unfolding the mysteries of the grand challenges, such as the dynamics of the global environment; the mechanism of DNA (deoxyribonucleic acid) sequencing; computer design of drugs to cope with deadly viruses; and computer simulation of future electronic materials, structures, and devices. Even though there are certain problems that computers cannot solve, as pointed out by Harel (2000), and hardware and software failures can be fatal, the human minds behind computers are nevertheless unlimited. Computers will never replace human beings in this regard and the quest for a better understanding of Nature will go on no matter how difficult the journey is. Computers will certainly help to make that journey more colorful and pleasant.

1.3 Computer algorithms and languages

Before we can use a computer to solve a specific problem, we must instruct the computer to follow certain procedures and to carry out the desired computational task. The process involves two steps. First, we need to transform the problem, typically in the form of an equation, into a set of logical steps that a computer can follow; second, we need to inform the computer to complete these logical steps.

Computer algorithms

The complete set of the logical steps for a specific computational problem is called a *computer* or *numerical algorithm*. Some popular numerical algorithms can be traced back over a 100 years. For example, Carl Friedrich Gauss (1866) published an article on the FFT (fast Fourier transform) algorithm (Goldstine, 1977,

pp. 249–53). Of course, Gauss could not have envisioned having his algorithm realized on a computer.

Let us use a very simple and familiar example in physics to illustrate how a typical numerical algorithm is constructed. Assume that a particle of mass m is confined to move along the x axis under a force $f(x)$. If we describe its motion with Newton's equation, we have

$$f = ma = m\frac{dv}{dt}, \tag{1.4}$$

where a and v are the acceleration and velocity of the particle, respectively, and t is the time. If we divide the time into small, equal intervals $\tau = t_{i+1} - t_i$, we know from elementary physics that the velocity at time t_i is approximately given by the average velocity in the time interval $[t_i, t_{i+1}]$,

$$v_i \simeq \frac{x_{i+1} - x_i}{t_{i+1} - t_i} = \frac{x_{i+1} - x_i}{\tau}; \tag{1.5}$$

the corresponding acceleration is approximately given by the average acceleration in the same time interval,

$$a_i \simeq \frac{v_{i+1} - v_i}{t_{i+1} - t_i} = \frac{v_{i+1} - v_i}{\tau}, \tag{1.6}$$

as long as τ is small enough. The simplest algorithm for finding the position and velocity of the particle at time t_{i+1} from the corresponding quantities at time t_i is obtained after combining Eqs. (1.4), (1.5), and (1.6), and we have

$$x_{i+1} = x_i + \tau v_i, \tag{1.7}$$

$$v_{i+1} = v_i + \frac{\tau}{m} f_i, \tag{1.8}$$

where $f_i = f(x_i)$. If the initial position and velocity of the particle are given and the corresponding quantities at some later time are sought (the initial-value problem), we can obtain them recursively from the algorithm given in Eqs. (1.7) and (1.8). This algorithm is commonly known as the Euler method for the initial-value problem. This simple example illustrates how most algorithms are constructed. First, physical equations are transformed into discrete forms, namely, difference equations. Then the desired physical quantities or solutions of the equations at different variable points are given in a recursive manner with the quantities at a later point expressed in terms of the quantities from earlier points. In the above example, the position and velocity of the particle at t_{i+1} are given by the position and velocity at t_i, provided that the force at any position is explicitly given by a function of the position. Note that the above way of constructing an algorithm is not limited to one-dimensional or single-particle problems. In fact, we can immediately generalize this algorithm to two-dimensional and three-dimensional problems, or to the problems involving more than one particle, such as the

motion of a projectile or a system of three charged particles. The generalized version of the above algorithm is

$$\mathbf{R}_{i+1} = \mathbf{R}_i + \tau \mathbf{V}_i, \tag{1.9}$$
$$\mathbf{V}_{i+1} = \mathbf{V}_i + \tau \mathbf{A}_i, \tag{1.10}$$

where $\mathbf{R} = (\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_n)$ is the position vector of all the n particles in the system; $\mathbf{V} = (\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n)$ and $\mathbf{A} = (\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_n)$, with $\mathbf{a}_j = \mathbf{f}_j / m_j$ for $j = 1, 2, \ldots, n$, are the corresponding velocity and acceleration vectors, respectively.

From a theoretical point of view, the *Turing machine* is an abstract representation of a *universal computer* and also a device to autopsy any algorithm. The concept was introduced by Alan Turing (1936–7) with a description of the universal computer that consists of a read and write head and a tape with an infinite number of units of binaries (0 or 1). The machine is in a specified state for a given moment of operation and follows instructions prescribed by a finite table. A computer algorithm is a set of logical steps that can be achieved by the Turing machine. Logical steps that cannot be achieved by the Turing machine belong to the class of problems that are not solvable by computers. Some such unsolvable problems are discussed by Harel (2000).

The logical steps in an algorithm can be sequential, parallel, or iterative (implicit). How to utilize the properties of a given problem in constructing a fast and accurate algorithm is a very important issue in computational science. It is hoped that the examples discussed in this book will help students learn how to establish efficient and accurate algorithms as well as how to write clean and structured computer programs for most problems encountered in physics and related fields.

Computer languages

Computer programs are the means through which we communicate with computers. The very first computer program was written by Ada Byron, the Countess of Lovelace, and was intended for the analytical engine proposed by Babbage in the mid-1840s. To honor her achievement, an object-oriented programming language (Ada), initially developed by the US military, is named after her. A *computer program* or *code* is a collection of statements, typically written in a well-defined computer *programming language*. Programming languages can be divided into two major categories: low-level languages designed to work with the given hardware, and high-level languages that are not related to any specific hardware.

Simple machine languages and assembly languages were the only ones available before the development of high-level languages. A machine language is typically in binary form and is designed to work with the unique hardware of a computer. For example, a statement, such as adding or multiplying two integers, is represented by one or several binary strings that the computer can recognize and follow. This is very efficient from computer's point of view, but extremely

labor-intensive from that of a programmer. To remember all the binary strings for all the statements is a nontrivial task and to debug a program in binaries is a formidable task. Soon after the invention of the digital computer, assembly languages were introduced to increase the efficiency of programming and debugging. They are more advanced than machine languages because they have adopted symbolic addresses. But they are still related to a certain architecture and wiring of the system. A translating device called an assembler is needed to convert an assembly code into a native machine code before a computer can recognize the instructions. Machine languages and assembly languages do not have portability; a program written for one kind of computers could never be used on others.

The solution to such a problem is clearly desirable. We need high-level languages that are not associated with the unique hardware of a computer and that can work on all computers. Ideal programming languages would be those that are very concise but also close to the logic of human languages. Many high-level programming languages are now available, and the choice of using a specific programming language on a given computer is more or less a matter of personal taste. Most high-level languages function similarly. However, for a researcher who is working at the cutting edge of scientific computing, the speed and capacity of a computing system, including the efficiency of the language involved, become critical.

A modern computer program conveys the tasks of an algorithm for a computational problem to a computer. The program cannot be executed by the computer before it is translated into the native machine code. A translator, a program called a *compiler*, is used to translate (or compile) the program to produce an executable file in binaries. Most compilers also have an option to produce an objective file first and then link it with other objective files and library routines to produce a combined executable file. The compiler is able to detect most errors introduced during programming, that is, the process of writing a program in a high-level language. After running the executable program, the computer will output the result as instructed.

The newest programming language that has made a major impact in the last few years is Java, an object-oriented, interpreted language. The strength of Java lies in its ability to work with web browsers, its comprehensive GUI (graphical user interface), and its built-in security and network support. Java is a truly universal language because it is fully platform-independent: "write once, run everywhere" is the motto that Sun Microsystems uses to qualify all the features in Java. Both the source code and the compiled code can run on any computer that has Java installed with exactly the same result. The Java compiler converts the source code (`file.java`) into a bytecode (`file.class`), which contains instructions in fixed-length byte strings and can be interpreted/executed on any computer under the Java interpreter, called JVM (Java Virtual Machine).

There are many advantages in Java, and its speed in scientific programming has been steadily increased over the last few years. At the moment, a carefully written Java program, combined with static analysis, just-in-time compiling, and