1 Types and sources of numerical error

1.1 Introduction

The job of a biomedical engineer often involves the task of formulating and solving mathematical equations that define, for example, the design criteria of biomedical equipment or a prosthetic organ or physiological/pathological processes occurring in the human body. Mathematics and engineering are inextricably linked. The types of equations that one may come across in various fields of engineering vary widely, but can be broadly categorized as: linear equations in one variable, linear equations with respect to multiple variables, nonlinear equations in one or more variables, linear and nonlinear ordinary differential equations, higher order differential equations of nth order, and integral equations. Not all mathematical equations are amenable to an analytical solution, i.e. a solution that gives an exact answer either as a number or as some function of the variables that define the problem. For example, the analytical solution for

(1) $x^2 + 2x + 1 = 0$ is $x = \pm 1$, and
(2) $dy/dx + 3x = 5$, with initial conditions $x = 0, y = 0$, is $y = 5x - 3x^2/2$.

Sometimes the analytical solution to a system of equations may be exceedingly difficult and time-consuming to obtain, or once obtained may be too complicated to provide insight.

The need to obtain a solution to these otherwise unsolvable problems in a reasonable amount of time and with the resources at hand has led to the development of **numerical methods**. Such methods are used to determine an approximation to the actual solution within some tolerable degree of error. A numerical method is an iterative mathematical procedure that can be applied to only certain types or forms of a mathematical equation, and under usual circumstances allows the solution to converge to a final value with a pre-determined level of accuracy or tolerance. Numerical methods can often provide exceedingly accurate solutions for the problem under consideration. However, keep in mind that the solutions are rarely ever exact. A closely related branch of mathematics is **numerical analysis**, which goes hand-in-hand with the development and application of numerical methods. This related field of study is concerned with analyzing the performance characteristics of established numerical methods, i.e. how quickly the numerical technique converges to the final solution and accuracy limitations. It is important to have, at the least, basic knowledge of numerical analysis so that you can make an informed decision when choosing a technique for solving a numerical problem. The accuracy and precision of the numerical solution obtained is dependent on a number of factors, which include the choice of the numerical technique and the implementation of the technique chosen.

Errors can creep into any mathematical solution or statistical analysis in several ways. Human mistakes include, for example, (1) entering incorrect data into

a computer, (2) errors in the mathematical expressions that define the problem, or (3) bugs in the computer program written to solve the engineering or math problem, which can result from logical errors in the code. A source of error that we have less control over is the quality of the data. Most often, scientific or engineering data available for use are imperfect, i.e. the true values of the variables cannot be determined with complete certainty. Uncertainty in physical or experimental data is often the result of imperfections in the experimental measuring devices, inability to reproduce exactly the same experimental conditions and outcomes each time, the limited size of sample available for determining the average behavior of a population, presence of a bias in the sample chosen for predicting the properties of the population, and inherent variability in biological data. All these errors may to some extent be avoided, corrected, or estimated using statistical methods such as confidence intervals. Additional errors in the solution can also stem from inaccuracies in the mathematical model. The model equations themselves may be simplifications of actual phenomena or processes being mimicked, and the parameters used in the model may be approximate at best.

Even if all errors derived from the sources listed above are somehow eliminated, we will still find other errors in the solution, called **numerical errors**, that arise when using numerical methods and electronic computational devices to perform numerical computations. These are actually unavoidable! **Numerical errors**, which can be broadly classified into two categories – round-off errors and truncation errors – are an integral part of these methods of solution and preclude the attainment of an exact solution. The source of these errors lies in the fundamental approximations and/or simplifications that are made in the representation of numbers as well as in the mathematical expressions that formulate the numerical problem. Any computing device you use to perform calculations follows a specific method to store numbers in a memory in order to operate upon them. Real numbers, such as fractions, are stored in the computer memory in floating-point format using the binary number system, and cannot always be stored with exact precision. This limitation, coupled with the finite memory available, leads to what is known as round-off error. Even if the numerical method yields a highly accurate solution, the computer round-off error will pose a limit to the final accuracy that can be achieved.

You should familiarize yourself with the types of errors that limit the precision and accuracy of the final solution. By doing so, you will be well-equipped to (1) estimate the magnitude of the error inherent in your chosen numerical method, (2) choose the most appropriate method for solution, and (3) prudently implement the algorithm of the numerical technique.

The origin of round-off error is best illustrated by examining how numbers are stored by computers. In Section 1.2, we look closely at the floating-point representation method for storing numbers and the inherent limitations in numeric precision and accuracy as a result of using binary representation of decimal numbers and finite memory resources. Section 1.3 discusses methods to assess the accuracy of estimated or measured values. The accuracy of any measured value is conveyed by the number of significant digits it has. The method to calculate the number of significant digits is covered in Section 1.4. Arithmetic operations performed by computers also generate round-off errors. While many round-off errors are too small to be of significance, certain floating-point operations can produce large and unacceptable errors in the result and should be avoided when possible. In Section 1.5, strategies to prevent the inadvertent generation of large round-off errors are discussed. The origin of truncation error is examined in Section 1.6. In Section 1.7 we introduce useful termination

Box 1.1A Oxygen transport in skeletal muscle

Oxygen is required by all cells to perform respiration and thereby produce energy in the form of ATP to sustain cellular processes. Partial oxidation of glucose (energy source) occurs in the cytoplasm of the cell by an anaerobic process called glycolysis that produces pyruvate. Complete oxidation of pyruvate to CO_2 and H_2O occurs in the mitochondria, and is accompanied by the production of large amounts of ATP. If cells are temporarily deprived of an adequate oxygen supply, a condition called hypoxia develops. In this situation, pyruvate no longer undergoes conversion in the mitochondria and is instead converted to lactic acid within the cytoplasm itself. Prolonged oxygen starvation of cells leads to cell death, called necrosis.

The circulatory system and the specialized oxygen carrier, hemoglobin, cater to the metabolic needs of the vast number of cells in the body. Tissues in the body are extensively perfused with tiny blood vessels in order to enable efficient and timely oxygen transport. The oxygen released from the red blood cells flowing through the capillaries diffuses through the blood vessel membrane and enters into the tissue region. The driving force for oxygen diffusion is the oxygen concentration gradient at the vessel wall and within the tissues. The oxygen consumption by the cells in the tissues depletes the oxygen content in the tissue, and therefore the oxygen concentration is always lower in the tissues as compared to its concentration in arterial blood (except when O_2 partial pressure in the air is abnormally low, such as at high altitudes). During times of strenuous activity of the muscles, when oxygen demand is greatest, the O_2 concentrations in the muscle tissue are the lowest. At these times it is critical for oxygen transport to the skeletal tissue to be as efficient as possible.

The skeletal muscle tissue sandwiched between two capillaries can be modeled as a slab of length L (see Figure 1.1). Let $N(x, t)$ be the O_2 concentration in the tissue, where $0 \leq x \leq L$ is the distance along the muscle length and t is the time. Let D be the O_2 diffusivity in the tissue and let Γ be the volumetric rate of O_2 consumption within the tissue.

Performing an O_2 balance over the tissue produces the following partial differential equation:

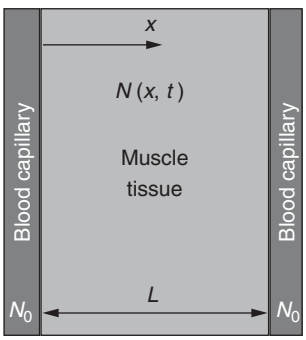
$$\frac{\partial N}{\partial t} = D\frac{\partial^2 N}{\partial x^2} - \Gamma.$$

The boundary conditions for the problem are fixed at $N(0, t) = N(L, t) = N_0$, where N_0 is the supply concentration of O_2 at the capillary wall. For this problem, it is assumed that the resistance to transport of O_2 posed by the vessel wall is small enough to be neglected. We also neglect the change in O_2 concentration along the length of the capillaries. The steady state or long-term O_2 distribution in the tissue is governed by the ordinary differential equation

$$D\frac{\partial^2 N}{\partial x^2} = \Gamma,$$

Figure 1.1

Schematic of O_2 transport in skeletal muscle tissue.



whose solution is given by

$$N_s = N_0 - \frac{\Gamma x}{2D}(L - x).$$

Initially, the muscles are at rest, consuming only a small quantity of O_2, characterized by a volumetric O_2 consumption rate Γ_1. Accordingly, the initial O_2 distribution in the tissue is $N_0 - \frac{\Gamma_1 x}{2D}(L - x)$.

Now the muscles enter into a state of heavy activity characterized by a volumetric O_2 consumption rate Γ_2. The time-dependent O_2 distribution is given by a Fourier series solution to the above partial differential equation:

$$N = N_0 - \frac{\Gamma_2 x}{2D}(L - x) + \frac{4(\Gamma_2 - \Gamma_1)L^2}{D} \sum_{n=1,\ n \text{ is odd}}^{\infty} \left[\frac{1}{(n\pi)^3} e^{-(n\pi)^2 Dt/L^2} \sin\left(\frac{n\pi x}{L}\right) \right].$$

In Section 1.6 we investigate the truncation error involved when arriving at a solution to the O_2 distribution in muscle tissue.

criteria for numerical iterative procedures. The use of robust convergence criteria is essential to obtain reliable results.

1.2 Representation of floating-point numbers

The arithmetic calculations performed when solving problems in algebra and calculus produce exact results that are mathematically sound, such as:

$$\frac{2.5}{0.5} = 5,\ \frac{\sqrt[3]{8}}{\sqrt{4}} = 1, \text{and } \frac{d}{dx}\sqrt{x} = \frac{1}{2\sqrt{x}}.$$

Computations made by a computer or a calculator produce a true result for any integer manipulation, but have less than perfect precision when handling real numbers. It is important at this juncture to define the meaning of "precision." Numeric **precision** is defined as the exactness with which the value of a numerical estimate is known. For example, if the true value of $\sqrt{4}$ is 2 and the computed solution is 2.0001, then the computed value is precise to within the first four figures or four significant digits. We discuss the concept of significant digits and the method of calculating the number of significant digits in a number in Section 1.4.

Arithmetic calculations that are performed by retaining mathematical symbols, such as $1/3$ or $\sqrt{7}$, produce exact solutions and are called symbolic computations. Numerical computations are not as precise as symbolic computations since numerical computations involve the conversion of fractional numbers and irrational numbers to their respective numerical or *digital* representations, such as $1/3 \sim 0.33333$ or $\sqrt{7} \sim 2.64575$. Numerical representation of numbers uses a finite number of digits to denote values that may possibly require an infinite number of digits and are, therefore, often inexact or approximate. The precision of the computed value is equal to the number of digits in the numerical representation that tally with the true digital value. Thus 0.33333 has a precision of five significant digits when compared to the true value of $1/3$, which is also written as $0.\overline{3}$. Here, the overbar indicates infinite repetition of the underlying digit(s). Calculations using the digitized format to represent all real numbers are termed as **floating-point arithmetic**. The format

Box 1.2 Accuracy versus precision: blood pressure measurements

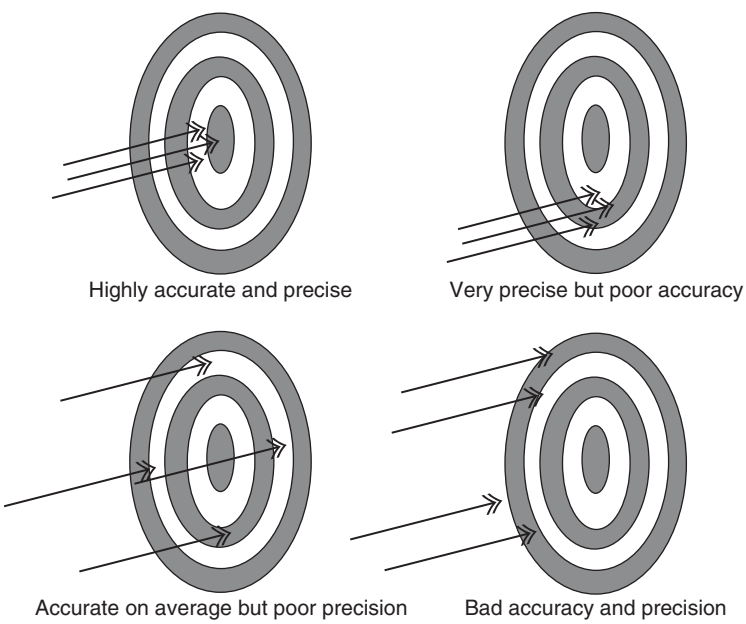
It is important that we contrast the two terms *accuracy* and *precision*, which are often confused. **Accuracy** measures how close the estimate of a measured variable or an observation is to its true value. **Precision** is the range or spread of the values obtained when repeated measurements are made of the same variable. A narrow spread of values indicates good precision.

For example, a digital sphygmomanometer consistently provides three readings of the systolic/diastolic blood pressure of a patient as 120/80 mm Hg. If the true blood pressure of the patient at the time the measurement was made is 110/70 mm Hg, then the instrument is said to be very precise, since it provided similar readings every time with few fluctuations. The three instrument readings are "on the same mark" every time. However, the readings are all inaccurate since the "correct target or mark" is not 120/80 mm Hg, but is 110/70 mm Hg.

An intuitive example commonly used to demonstrate the difference between accuracy and precision is the bulls-eye target (see Figure 1.2).

Figure 1.2

The bulls-eye target demonstrates the difference between accuracy and precision.



standards for **floating-point number representation** by computers are specified by the IEEE Standard 754. The binary or base-2 system is used by digital devices for storing numbers and performing arithmetic operations. The binary system cannot precisely represent all *rational numbers* in the decimal or base-10 system. The imprecision or error inherent in computing when using floating-point number representation is called **round-off error**. Round-off thus occurs when real numbers must be approximated using a limited number of significant digits. Once a round-off error is introduced in a floating-point calculation, it is carried over in all subsequent computations. However, round-off is of fundamental advantage to the efficiency of performing computations. Using a fixed and finite number of digits to represent

each number ensures a reasonable speed in computation and economical use of the computer memory.

Round-off error results from a trade-off between the efficient use of computer memory and accuracy.

Decimal floating-point numbers are represented by a computer in standardized format as shown below:

$$\pm 0.f_1 f_2 f_3 f_4 f_5 f_6 \ldots f_{s-1} f_s \times 10^k,$$

significand — 10 raised to the power k

where f is a decimal digit from 0 to 9, s is the number of significant digits, i.e. the number of digits in the significand as dictated by the precision limit of the computer, and k is the exponent. The advantage of this numeric representation scheme is that the **range** of representable numbers (as determined by the largest and smallest values of k) can be separated from the degree of **precision** (which is determined by the number of digits in the significand). The power or exponent k indicates the **order of magnitude** of the number. The notation for the order of magnitude of a number is $O(10^k)$. Section 1.7 discusses the topic of "estimation of order of magnitude" in more detail.

This method of numeric representation provides the best approximation possible of a real number within the limit of s significant digits. The real number may, however, require an infinite number of significant digits to denote the true value with perfect precision. The value of the last significant digit of a *floating-point number* is determined by one of two methods.

(1) **Truncation** Here, the numeric value of the digit f_{s+1} is not considered. The value of the digit f_s is unaffected by the numeric value of f_{s+1}. The floating-point number with s significant digits is obtained by dropping the $(s + 1)$th digit and all digits to its right.

(2) **Rounding** If the value of the last significant digit f_s depends on the value of the digits being discarded from the floating-point number, then this method of numeric representation is called rounding. The generally accepted convention for rounding is as follows (Scarborough, 1966):
   (a) if the numeric value of the digits being dropped is greater than five units of the f_{s+1}th position, then f_s is changed to $f_s + 1$;
   (b) if the numeric value of the digits being dropped is less than five units of the f_{s+1}th position, then f_s remains unchanged;
   (c) if the numeric value of the digits being dropped equals five units of the f_{s+1}th position, then
      (i) if f_s is even, f_s remains unchanged,
      (ii) if f_s is odd, f_s is replaced by $f_s + 1$.

This last convention is important since, on average, one would expect the occurrence of the f_s digit as odd only half the time. Accordingly, by leaving f_s unchanged approximately half the time when the f_{s+1} digit is exactly equal to five units, it is intuitive that the errors caused by rounding will to a large extent cancel each other.

As per the rules stated above, the following six numbers are rounded to retain only five digits:

$$\begin{aligned} 0.345903 &\rightarrow 0.34590, \\ 13.85748 &\rightarrow 13.857, \\ 7983.9394 &\rightarrow 7983.9, \\ 5.20495478 &\rightarrow 5.2050, \\ 8.94855 &\rightarrow 8.9486, \\ 9.48465 &\rightarrow 9.4846. \end{aligned}$$

The rounding method is used by computers to store floating-point numbers. Every machine number is precise to within $5 \times 10^{-(s+1)}$ units of the original real number, where s is the number of significant figures.

Using MATLAB

A MATLAB function called `round` is used to round numbers to their nearest integer. The number produced by `round` does not have a fractional part. This function rounds down numbers with fractional parts less than 0.5 and rounds up for fractional parts equal to or greater than 0.5. Try using `round` on the numbers 1.1, 1.5, 2.50, 2.49, and 2.51. Note that the MATLAB function `round` does not round numbers to s significant digits as discussed in the definition above for rounding.

1.2.1 How computers store numbers

Computers store all data and instructions in binary coded format or the **base-2** number system. The machine language used to instruct a computer to execute various commands and manipulate operands is written in binary format – a number system that contains only two digits: 0 and 1. Every binary number is thus constructed from these two digits only, as opposed to the base-10 number system that uses ten digits to represent numbers. The differences between these two number systems can be further understood by studying Table 1.1.

Each digit in the binary system is called a bit (binary digit). Because a bit can take on two values, either 0 or 1, each value can represent one of two physical states – on or off, i.e. the presence or absence of an electrical pulse, or the ability of a transistor to switch between the on and off states. Binary code is thus found to be a convenient method of encoding instructions and data since it has obvious physical significance and can be easily understood by the operations performed by a computer.

The range of the magnitude of numbers, as well as numeric precision that a computer can work with, depends on the number of bits allotted for representing numbers. Programming languages, such as Fortran and C, and mathematical software packages such as MATLAB allow users to work in both single and double precision.

1.2.2 Binary to decimal system

It is important to be familiar with the methods for converting numbers from one base to another in order to understand the inherent limitations of computers in working with real numbers in our base-10 system. Once you are well-versed in the

Table 1.1. *Equivalence of numbers in the decimal (base-10) and binary (base-2) systems*

Decimal system (base 10)	Binary system (base 2)	Conversion of binary number to decimal number
0	0	$0\times2^0=0$
1	1	$1\times2^0=1$
2	1 0	$1\times2^1+0\times2^0=2$
3	1 1	$1\times2^1+1\times2^0=3$
4	1 0 0	$1\times2^2+0\times2^1+0\times2^0=4$
5	1 0 1	$1\times2^2+0\times2^1+1\times2^0=5$
6	1 1 0	$1\times2^2+1\times2^1+0\times2^0=6$
7	1 1 1	$1\times2^2+1\times2^1+1\times2^0=7$
8	1 0 0 0	$1\times2^3+0\times2^2+0\times2^1+0\times2^0=8$
9	1 0 0 1	$1\times2^3+0\times2^2+0\times2^1+1\times2^0=9$
10	1 0 1 0	$1\times2^3+0\times2^2+1\times2^1+0\times2^0=10$
	↑ ↑ ↑ ↑	
	$\mathbf{2^3\ 2^2\ 2^1 2^0}$	
	binary position indicators	

ways in which round-off errors can arise in different situations, you will be able to devise suitable algorithms that are more likely to minimize round-off errors. First, let's consider the method of converting binary numbers to the decimal system. The *decimal number* 111 can be expanded to read as follows:

$$\begin{aligned}111 &= 1\times10^2+1\times10^1+1\times10^0\\ &= 100+10+1\\ &= 111.\end{aligned}$$

Thus, the position of a digit in any number specifies the magnitude of that particular digit as indicated by the power of 10 that multiplies it in the expression above. The first digit of this base-10 integer from the right is a multiple of 1, the second digit is a multiple of 10, and so on. On the other hand, if 111 is a binary number, then the same number is now equal to

$$\begin{aligned}111 &= 1\times2^2+1\times2^1+1\times2^0\\ &= 4+2+1\\ &= 7 \text{ in base 10.}\end{aligned}$$

The decimal equivalent of 111 is also provided in Table 1.1. In the binary system, the position of a binary digit in a binary number indicates to which power the multiplier, 2, is raised. Note that the largest decimal value of a binary number comprising n bits is equal to 2^n-1. For example, the binary number 11111 has 5 bits and is the binary equivalent of

$$\begin{aligned}11111 &= 1\times2^4+1\times2^3+1\times2^2+1\times2^1+1\times2^0\\ &= 16+8+4+2+1\\ &= 31\\ &= 2^5-1.\end{aligned}$$

What is the range of integers that a computer can represent? A certain fixed number of bits, such as 16, 32, or 64 bits, are allotted to represent every integer. This fixed maximum number of bits used for storing an integer value is determined by the computer hardware architecture. If 16 bits are used to store each integer value in binary form, then the maximum integer value that the computer can represent is $2^{16}-1 = 65\,535$. To include representation of negative integer values, 32 768 is subtracted internally from the integer value represented by the 16-bit number to allow representation of integers in the range of [−32 768, 32 767].

What if we have a fractional binary number such as 1011.011 and wish to convert this binary value to the base-10 system? Just as a digit to the right of a radix point (decimal point) in the base-10 system represents a multiple of 1/10 raised to a power depending on the position or place value of the decimal digit with respect to the decimal point, similarly a binary digit placed to the right of a radix point (binary point) represents a multiple of 1/2 raised to some power that depends on the position of the binary digit with respect to the radix point. In other words, just as the fractional part of a decimal number can be expressed as a sum of the negative powers of 10 (or positive powers of 1/10), similarly a binary number fraction is actually the sum of the negative powers of 2 (or positive powers of 1/2). Thus, the decimal value of the binary number 1011.011 is calculated as follows:

$$\begin{aligned}&(1\times 2^3)+(0\times 2^2)+(1\times 2^1)+(1\times 2^0)+(0\times (1/2)^1)\\&\quad+(1\times (1/2)^2)+(1\times (1/2)^3)\\&\quad=8+2+1+0.25+0.125\\&\quad=11.375.\end{aligned}$$

1.2.3 Decimal to binary system

Now let's tackle the method of converting decimal numbers to the base-2 system. Let's start with an easy example that involves the conversion of a decimal integer, say 123, to a binary number. This is simply done by resolving the integer 123 as a series of powers of 2, i.e. $123 = 2^6 + 2^5 + 2^4 + 2^3 + 2^1 + 2^0 = 64 + 32 + 16 + 8 + 2 + 1$. The powers to which 2 is raised in the expression indicate the positions for the binary digit 1. Thus, the binary number equivalent to the decimal value 123 is 1111011, which requires 7 bits. This expansion process of a decimal number into the sum of powers of 2 is tedious for large decimal numbers. A simplified and straightforward procedure to convert a decimal number into its binary equivalent is shown below (Mathews and Fink, 2004).

We can express a positive base-10 integer I as an expansion of powers of 2, i.e.

$$I = b_n\times 2^n + b_{n-1}\times 2^{n-1} + \cdots + b_2\times 2^2 + b_1\times 2^1 + b_0\times 2^0,$$

where $b_0, b_1, \ldots, b_n$ are binary digits each of value 0 or 1. This expansion can be rewritten as follows:

$$I = 2(b_n\times 2^{n-1} + b_{n-1}\times 2^{n-2} + \cdots + b_2\times 2^1 + b_1\times 2^0) + b_0$$

or

$$I = 2\times I_1 + b_0,$$

where

$$I_1 = b_n \times 2^{n-1} + b_{n-1} \times 2^{n-2} + \cdots + b_2 \times 2^1 + b_1 \times 2^0.$$

By writing I in this fashion, we obtain b_0.
Similarly,

$$I_1 = 2(b_n \times 2^{n-2} + b_{n-1} \times 2^{n-3} + \cdots + b_2 \times 2^0) + b_1,$$

i.e.

$$I_1 = 2 \times I_2 + b_1,$$

from which we obtain b_1 and

$$I_2 = b_n \times 2^{n-2} + b_{n-1} \times 2^{n-3} + \cdots + b_2 \times 2^0.$$

Proceeding in this way we can easily obtain all the digits in the binary representation for I.

Example 1.1 Convert the integer 5089 in base-10 into its binary equivalent. Based on the preceding discussion, 5089 can be written as

$$5089 = 2 \times 2544 + 1 \rightarrow b_0 = 1$$

$$2544 = 2 \times 1272 + 0 \rightarrow b_1 = 0$$

$$1272 = 2 \times 636 + 0 \rightarrow b_2 = 0$$

$$636 = 2 \times 318 + 0 \rightarrow b_3 = 0$$

$$318 = 2 \times 159 + 0 \rightarrow b_4 = 0$$

$$159 = 2 \times 79 + 1 \rightarrow b_5 = 1$$

$$79 = 2 \times 39 + 1 \rightarrow b_6 = 1$$

$$39 = 2 \times 19 + 1 \rightarrow b_7 = 1$$

$$19 = 2 \times 9 + 1 \rightarrow b_8 = 1$$

$$9 = 2 \times 4 + 1 \rightarrow b_9 = 1$$

$$4 = 2 \times 2 + 0 \rightarrow b_{10} = 0$$

$$2 = 2 \times 1 + 0 \rightarrow b_{11} = 0$$

$$1 = 2 \times 0 + 1 \rightarrow b_{12} = 1$$

Thus the binary equivalent of 5089 has 13 binary digits and is 1001111100001. This algorithm, used to convert a decimal number into its binary equivalent, can be easily incorporated into a MATLAB program.

1.2.4 Binary representation of floating-point numbers

Floating-point numbers are numeric quantities that have a significand indicating the value of the number, which is multiplied by a base raised to some power. You are