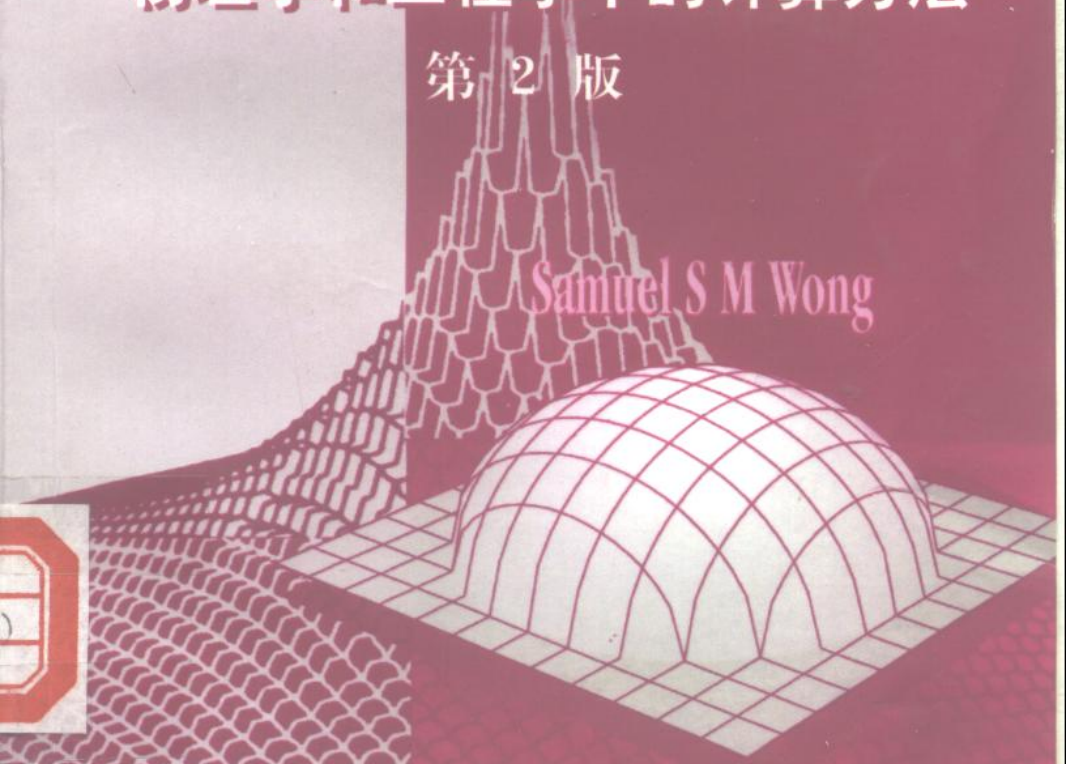
COMPUTATIONAL METHODS IN PHYSICS AND ENGINEERING

2nd edition

物理学和工程学中的计算方法

第 2 版

Samuel S M Wong



World Scientific
世界图书出版公司

COMPUTATIONAL METHODS IN PHYSICS AND ENGINEERING

2nd edition

Samuel S M Wong
University of Toronto

World Scientific
世界图书出版公司

书　　名：Computational Methods in Physics and Engineering 2nd ed.
作　　者：S.S.M.Wong
中 译 名：物理学和工程学中的计算方法　第 2 版
出 版 者：世界图书出版公司北京公司
印 刷 者：北京中西印刷厂
发　　行：世界图书出版公司北京公司（北京朝内大街 137 号　100010）
开　　本：1/32　850×1168　印　张：16.5
出版年代：2000 年 6 月
书　　号：ISBN 7-5062-4720-8/O・300
版权登记：图字 01-2000-0114
定　　价：60.00 元

Published by

World Scientific Publishing Co. Pte. Ltd.

P O Box 128, Farrer Road, Singapore 912805

USA office: Suite 1B, 1060 Main Street, River Edge, NJ 07661

UK office: 57 Shelton Street, Covent Garden, London WC2H 9HE

Library of Congress Cataloging-in-Publication Data

Wong, S. S. M. (Samuel Shaw Ming)

Computational methods in physics and engineering / Samuel S. M. Wong, -- 2nd ed.

p. cm.

Includes bibliographical references and index.

ISBN 9810230176 ISBN 9810230435 (pbk)

1. Physics -- Data processing. 2. Mathematical physics.
3. Engineering -- Data processing. 4. Engineering mathematics.
I. Title.

QC52.W66 1997

530'.0285--dc21 96-51106

CIP

British Library Cataloguing-in-Publication Data

A catalogue record for this book is available from the British Library.

First edition published in 1992 by Prentice-Hall, Inc.

This edition © 1977 by World Scientific Publishing Co. Pte. Ltd.

COMPUTATIONAL METHODS IN PHYSICS AND ENGINEERING

Preface to the First Edition

Computational methods form an increasingly important part of the undergraduate curriculum in physics and engineering these days. This book is mainly concerned with the ways that computers may be used to advance a student's understanding of physics. A large part of the material is common to engineering as well.

The subject matter covered in this volume may be classified also under the title of "computational physics." There are several ways to organize the material that should be included. The choice made here is to follow the traditional approach of mathematical physics. That is, the chapters and sections are grouped around methods, with physical problems used as the motivation and examples. One attractive alternative is to group around physical phenomena. The difficulty of following this way of organization is the heavy reliance on the physics background of the readers, thus making it harder to follow for students at early stages of their education. For this reason, such an approach is rejected.

The intimate relation between physics and mathematics may be seen by the way that physics is usually taught in the undergraduate curriculum. With a knowledge of calculus, for example, the subject of mechanics is discussed in a more rigorous manner. By the time the student is introduced to differential equations, topics such as harmonic oscillators and alternating current circuits, are brought in. Long experience in the community has shown this way of teaching to be very successful. The major problem here is the delay in introducing certain other basic concepts, because of the need to acquire first a certain maturity in mathematics. The fault is not with the mathematics required but the way it is used. For example, discussions on a pendulum are usually limited to small amplitudes at the early years. For finite amplitudes, the differential equation is nonlinear and the necessary skill to solve such equations by analytical methods comes only in later years. On the other hand, it is possible to use the same numerical methods to solve both types of differential equations for the pendulum problem and they are no more difficult than analytical methods. In this way, the discussion on pendulum does not have to be limited to small amplitude oscillations.

Until quite recently, the mathematics required in undergraduate physics and engineering, to a large extent, consists of analytical techniques to manipulate algebraic equations, to carry out integrals, and to solve simple differential equations. In addition to these "algebraic" methods, there is also a large class of numerical approaches that can be used to solve physical problems. While it is true that computers have made numerical calculations popular, many of the methods have their origins with the same group of mathematicians as the algebraic methods, such as Gauss and Newton. In the intervening years since the introduction of these mathematical techniques, numerical calculations have lost out to "algebraic" ones, perhaps because of the tedium of carrying out numerical calculations by hand. This reason is certainly no longer true, as attested by the explosion of numerical solutions in research papers. In spite of its popularity in research, the introduction of computational physics to the undergraduate syllabus is only starting.

To carry out numerical calculations on a computer, the usual practice is to write one or more programs in one of the high-level languages, such as FORTRAN or C. To simplify the process, one may make use of standard subroutine libraries to take over specific tasks, such as inverting a determinant or diagonalizing a matrix. In contrast, the general approach to algebraic computation is to make use of one of the symbolic manipulation packages. Most of these packages are very powerful and are able to carry out a large variety of complicated calculations. Although a substantial amount of "programming" can be done in most cases, the symbolic manipulation instructions are not programming languages on the same level as, for example, FORTRAN or C. Furthermore, there has been little attempt to standardize the instructions between different packages. As a result, any discussions of algebraic calculations in a volume on computational methods must either be very abstract or very specific in terms of one of these packages. The choice made here is the former, as it is not clear what is available to the average reader.

The results obtained from computer calculations often appear in the form of a large table of numbers. For human beings to comprehend such huge quantities of information, graphical presentations are essential. For this reason, graphical techniques are essential parts of computational methods. On the other hand, it is not possible to cover all three aspects, numerical, symbolic, and graphical, in a single volume. The choice made here is to select a few of the standard topics in physics covered in the undergraduate curriculum and present them in ways that computers may be useful for their solutions. Even this is too big a task. The compromise is to put the emphasis on numerical techniques. For reasons mentioned in the previous paragraph, only an introduction is made to symbolic manipulation techniques. Computer graphics is perhaps one of the fastest growing areas in computing. For this reason also, it is best to leave everything beyond an introduction to volumes specializing on the subject.

There is quite a bit of interest in the physics community in developing courses in computational physics, as has already been done in many places. Such courses should be regarded as in parallel with the more traditional ones in mathematical physics and experimental physics. It is one of the aims of this volume to serve as a textbook or major reference for such a course. At the same time, many graduate students and senior undergraduates may not have benefited from such a course. It is also the intention here to serve this group of physicists. Engineers and other professional people who make use of computational techniques in physics may also find it useful to examine some of the background involved in solving some of the physical problems on computers.

Although a set of computer program is present here, it is not the primary intention of the author to provide a library of subroutines for common problems in physics. The computer programs used as examples and included in the accompanying diskette are intended as illustrations for some of the materials discussed. They can be modified for other applications. However, before one does that, as with any computer program, the programs should be thoroughly tested first for the intended purpose.

S. S. M. Wong

Preface to the Second Edition

The excellent reception of the first edition by the reader community worldwide makes it imperative for a new edition. Many subtle changes have taken place in scientific computation in the time interval. Most of these are driven by the tremendous increase in computational power available to the individual user, making it possible to carry out work almost unthinkable a little while ago and there is every indication that the trend will continue in the near future. As a result, computational techniques become even more indispensable to engineers and scientists than ever before. At the same time, the convenience of communication through internet is also having an impact on how computers are used and how some of the computations are carried out these days.

The main changes in the second edition include the addition of a chapter on finite element methods. The growth of available computing power makes it more and more attractive to solve many complicated differential equations numerically and a different approach from the traditional finite difference methods is getting increasing attention, especially in the physics community. In order to keep the volume from expanding into unmanageable size, some sacrifices have to be made. These include introductory chapters on graphics and computer algebra and a couple of sections on topics that are less popular. Some derivations that can be left to references are also omitted. In order to accommodate these changes, most of chapters have been substantially rewritten. The end result is a volume that is more ideal in size both for the classroom and on the desk.

Many valuable suggestions have been received from readers and they have been incorporated, as far as possible, into the new edition. The author is greatly indebted to colleagues for caring to communicate and share their thoughts. The strong encouragement and support from World Scientific also form an essential ingredient in making the decision to finish on the second edition.

Samuel S.M. Wong
Toronto

Contents

Chapter 1

Computational Methods

Modern electronic computers owe their origin, to a large extent, to the needs in science and engineering. In the 50 years or so since their appearance, computers have out-performed their original goals of solving numerical problems and keeping tracks of information. They are now an essential tool in almost every aspect of the daily routines of engineers and scientists, from data collection to writing technical reports. The high speed of computation available to us these days opens up not only new ways of carrying out traditional tasks but also new areas of endeavor that have implications going well beyond what we can realize at the moment. Our concern here is limited to a small, albeit important, corner of the role of modern computers in science and engineering, namely some of the general techniques to solve common problems encountered in physics and engineering.

1-1 Numerical calculations and beyond

When we use a computer to solve a problem in science, the general assumption is that it is done numerically. Indeed, the proper name of most computers in the market is "digital computer," reminding us of the fact that numbers are being manipulated. However, in addition to mathematical operations, such as addition and multiplication, the central processor of a computer is also capable of logical operations, that is, making decisions depending on whether a particular condition is true or false. Furthermore, a binary digit, or "bit," of the computer memory may be regarded as a logical unit, representing the value "true" if it is on (= 1) and "false" if it is off (= 0). In this way, a computer can be programmed equally well to carry out logical decisions or, more generally, symbolic manipulations.

At the same time, the computer screen is made of lines of horizontal dots or "pixels." On a monochrome screen, each dot can be turned on or off. For a color monitor, there is the further capability of displaying different colors at each dot. As a result, very effective graphical images can be displayed. The same is also true for paper output. For this reason, in addition to numerical calculation and symbolic manipulation, computers are used extensively for graphics.

In this volume, we shall be mainly concerned with numerical calculations. Before we get totally immersed in the topic, it is useful to remind ourselves that both symbolic manipulation and graphic presentation are also important in scientific applications of computers. We shall give a brief description of both topics before getting on to numerical calculations.

Computer algebra Among the various possibilities of using computers for "artificial intelligence" applications, symbolic manipulation, more commonly referred to as computer algebra, is an important tool in scientific endeavors. However, we shall not go into the subject in this volume. The main reason for this choice is that, most computer algebra are carried out using one of the available "packages," such as Maple (Symbolic Computation Group, University of Waterloo), Mathematica (Wolfram Research, Inc), and Reduce (Rand Corporation). Although the Lisp language, for example, is designed for the purpose of symbolic manipulation, most of computer algebra applications these days are carried without explicitly going to the level of actually programming a computer using one of the general purpose programming languages.

In most computer applications, it is desirable for us to give the instructions in a language that is as close as possible to the working language of the subject we are involved with. In the case of algebraic calculations, it is natural for us to want to work in terms of algebraic equations. For example, if we wish to solve the set of equations:

$$\begin{aligned} ax + by &= c \\ dx + ey &= f \end{aligned} \tag{1-1}$$

it would be nice if all we need to say to the computer is something like

SOLVE

$$ax + by = c$$
$$dx + ey = f$$

FOR x AND y.

However, to solve Eq. (1-1) requires some knowledge of linear algebra that is beyond the basic mathematical and logical operations a computer is designed for.

As we shall see later in §5-1, the solution may be expressed in terms of ratios of determinants. In general, we cannot expect a computer or any general purpose programming language to possess such advanced knowledge of algebra. On the other hand, for commonly encountered applications, one can think of developing a set of codes that specialize in these problems. In this way, we can always call up the codes whenever we want to solve, for example, a set of linear equations as in the example above. In fact, we can put together a number of such codes that carry out related functions, such as evaluating determinants, inverting matrices, finding the roots of linear equations, and write a "driver" to manage them. Such a collection is often loosely referred to as a "package." The development of computer algebra, to a large

extent, has followed this route. As a result, algebraic calculations are usually carried on computers in terms of one of the existing packages. In fact, some of these symbolic manipulation packages are so versatile that they can be regarded as programming languages for a large class of calculations.

Because of this tendency, discussions on algebraic calculations are often based on one of the popular packages. Since most textbooks in physics and engineering have already done a good job in presenting the algebraic aspects of various topics, the main work we need to do is to cast the problems in terms of the language of one of the packages. We shall not do this here. Instead, we shall give an example of algebraic calculation later in §1-4, just to provide some introduction to those who have not been exposed to the wonders of computer algebra.

The development of numerical methods for physics and engineering is somewhat different. Although there are many excellent "packages" available to carry out specific calculations, such as eigenvalue problems and matrices,[2] they tend to exist in the form of subprogram libraries. In this case, the user often has to write a "calling" program to transform the problem in hand into one that can take advantage of the library to perform some of the calculations. Similar to other tools, a basic knowledge of the numerical methods used in these subprogram is essential in this case.

Computer graphics An important market for computers these days is in graphics. This is, in part, due to the success in computer animation and computer aided design. Such applications clearly belong to totally different treatments from what we intend here. However, there is a small area of computer graphics that is important to numerical work, namely graphical representation of results.

In numerical calculations, the results often appear as a table of numbers and, for a complicated problem, such a table can be an extensive one. A good way to gain an overall feeling for a large set of numbers is to view them in the form of a plot. Before computers, plots are usually done on a sheet of graph paper. For simplicity, let us consider the problem of making a linear plot for some function y of a single independent variable x. The graph paper we shall use in this case is nothing but a sheet of paper with $(N_x + 1)$ evenly spaced horizontal lines and $(N_y + 1)$ evenly spaced vertical lines. The plotting area of our graph paper may therefore be regarded as made up of $N_x \times N_y$ squares. We can select one of the horizontal lines as our x-axis and a vertical line as the y-axis. The scales of our two axes are set by the ranges of values we wish to display. The actual plotting is carried out by putting on the graph paper a symbol for the value of y corresponding to each one of the values of x we are interested in. For a continuous function, the plot of y versus x is a continuous curve. However, for our purpose here, such a continuous curve may be regarded as a collection of closely spaced points, one for each possible values of the independent variable x.

We can follow basically the same steps to plot the graph on a computer screen or a sheet of output. The reason is that both types of device are made of a number of dots or pixels, very similar to the squares on our graph paper. For example, many

monitors are said to have a resolution of 1024×746. For our present purpose, it may be regarded as a graph paper with 1024 horizontal lines and 746 vertical lines. The only difference is that, for a variety of technical reasons which we do not need to go into here, the spacings between the horizontal lines and vertical lines are not necessarily equal. In fact, the aspect ratio, i.e., the ratio of the horizontal and vertical size of each element, on a computer screen is usually less than one. As a result, each one of the basic elements on our "graph paper" is, often, a rectangle instead of a square as on a normal sheet of graph paper. While this difference causes some nuisance in displaying a graph, it does not impose any fundamental problem and we shall ignore it here. For output on paper, it is not difficult to obtain resolutions of 300 to 600 dots per inch. This means that we can easily have the equivalent of 300^2 to 600^2 elements on each square inch of a sheet of output.

The computer screen (often paper output as well) has the advantage of color. As a result, we have an additional dimension to express our "graphs" that is not easily available on graph papers. Furthermore, it is possible to generate many frames of a "graph" in a short time and, as a result, one can have a "movie" of our calculated results to show, for example, the time development of a process.

Although the basic principles of computer graphics are simple, the actual applications require some preparations. Most computers come with machine-level instructions to access and to manipulate the pixels. However, it will be extremely tedious to plot a graph using such low-level instructions. In fact, what we prefer is that, for example, once two arrays, say X_ARRAY and Y_ARRAY, are generated, we can issue an instruction like

```
PLOT Y_ARRAY versus X_ARRAY.
```

The computer will then go and find the best axes and scales to represent y as a function of x and display the results on the screen. Another instruction,

```
OUTPUT PLOT
```

will produce a printed version of the plot on a sheet of paper. For more complicated plots, such as histograms, log plots, contours, and surfaces, we can think of, at least in principle, developing similar conversation-like instructions.

Needless to say, we are not close to this ideal level of graphics programming on computers. Partly because of the fact that the standardization of computer graphics hardware and software is still in the development stage, there are only, at the time of writing, the beginnings of common high-level graphics programming languages and "interfaces" that are portable between different types of computers. As a result, we must once again resort to "packages." Similar to computer algebra, there are many fairly extensive plotting packages available, both public domain and commercial, and most plottings are done using one of these packages.

1-2 Integers and floating numbers

Most computers make a clear distinction between integers and floating numbers. An integer is a number, such as 5, −213, and 0, without a part that must be represented by a decimal point. A floating or *real* number is any other type of number, such as $3.1415926\ldots$, 3.0×10^{23}, and -9.9, that requires a decimal point to specify its value. The reason for differentiating between these two types of numbers comes from the structure of the computer memory.

The basic unit for storing a number in a computer is a *bit*, the state of an electronic component that is either on or off, as we saw earlier. The two possible states of a bit may be used to represent two numerical values 0 (off) and 1 (on). Since a single bit is too small for most interests, 8 bits are grouped together into a *byte*. The status of a byte may be represented by an eight-digit binary number $b\,\square\,\square\,\square\,\square\,\square\,\square\,\square\,\square$, where we have added a prefix b in front of the number to indicate that it is in the binary representation. For example, the integer 5 is shown as $b00000101$ or simply as $b101$. The largest integer that can be represented in one byte of storage is then $b11111111 = 2^8 - 1 = 255$.

Before we leave the subject of internal representation of integers in the computer memory, we shall define two other representations. The binary representation used above is inconvenient in many cases because of the large number of digits required to express most of the integers of interest to us. The hexadecimal representation is based on powers of 16 (in an analogous way as the decimal system is based on powers of 10). Each hexadecimal digit can take on values 0 through 15, and they are usually written as $z0$, $z1$, $z2$, $z3$, $z4$, $z5$, $z6$, $z7$, $z8$, $z9$, zA, zB, zC, zD, zE, and zF, where we have added a prefix z to indicate that they are given in hexadecimal representation. In terms of computer memory, each hexadecimal digit represents one of the possible values stored in four binary digits ($2^4 = 16$). The value that can be stored in a byte is then represented by two hexadecimal digits. Examples of numbers in the hexadecimal representation and their corresponding values in decimal and binary representations are given in Table 1-1.

Another way of displaying binary-based numbers is the octal representation. To distinguish it from others, we shall prefix a number in the octal representation with the letter o. (The lowercase o is used here instead of the uppercase so as to make it easy to differentiate it from the number zero.) Each octal digit represents the value given by 3 bits. This is convenient for some models of computer whose memories are made of multiples of 3 bits, such as 36 and 60. In addition, octal representation is also a convenient way for carrying out many types of manipulations.

In many calculations, a byte, which can only store integers up to 255, is still too small. For this reason, each integer is often assigned either two or four bytes of memory. Such a grouping of bytes is sometimes called a computer word, or just *word* for short. Before we go into the question of the range of integer values a computer word can store, we must recall that most numbers we are interested in have a ± sign associated with them. It is common practice to designate the first bit of an integer

Table 1-1: Decimal, hexadecimal, octal, and binary representations of numbers.

Decimal	Hexa-decimal	Octal	Binary	Decimal	Hexa-decimal	Octal	Binary
0	*z*0	*o*0	*b*0	10	*z*A	*o*12	*b*1010
1	*z*1	*o*1	*b*1	11	*z*B	*o*13	*b*1011
2	*z*2	*o*2	*b*10	12	*z*C	*o*14	*b*1100
3	*z*3	*o*3	*b*11	13	*z*D	*o*15	*b*1101
4	*z*4	*o*4	*b*100	14	*z*E	*o*16	*b*1110
5	*z*5	*o*5	*b*101	15	*z*F	*o*17	*b*1111
6	*z*6	*o*6	*b*110	16	*z*10	*o*20	*b*1 0000
7	*z*7	*o*7	*b*111	17	*z*11	*o*21	*b*1 0001
8	*z*8	*o*10	*b*1000	18	*z*12	*o*22	*b*1 0010
9	*z*9	*o*11	*b*1001	19	*z*13	*o*23	*b*1 0011
255	*z*FF	*o*377	*b*1111 1111				
256	*z*100	*o*400	*b*1 0000 0000				
257	*z*101	*o*401	*b*1 0000 0001				

word as the sign bit. Thus, for a two-byte integer I_2, only 15 bits are available to store the magnitude of the number. The possible values that can be represented by such a word is then

$$-32,768 \leq I_2 \leq +32,767.$$

That is, -2^{15} through $(2^{15}-1)$. The reason that the maximum positive integer value is one less than 2^{15} comes from the fact that $+0$ must also be considered as one of the integers. (Why, then, is the maximum absolute value of a negative integer one larger?) For a four-byte word, the possible value of an integer is then

$$-2,147,483,648 \leq I_4 \leq +2,147,483,647$$

corresponding to -2^{31} to $(2^{31}-1)$. Although the allowed range of integer values for I_4 may be large, it is still not adequate for many purposes. For example, the upper limit of I_4 is less than $13! = 6,227,020,800$. As a result, it may not be possible to carry out certain types of calculations that involve factorials. We shall see later ways to circumvent this difficulty.

For most calculations in science and engineering, the use of integers alone is too restrictive. Floating numbers broaden the range of values that can be kept in the computer memory by allocating a part of a word to store the exponent of each number. That is, each number now has a sign, a fraction part or mantissa, and an exponent. Since the computer memory is made of bits, some arrangements must be made to represent a number in this way. The usual case is to use four bytes for a single-precision number. Among the 32 bits in such a word, 1 bit is devoted to the sign, 8 bits are assigned to the exponent, and the remaining 23 bits are left for the mantissa. In this way, numbers with absolute values in the range from approximately