LEE W. JOHNSON
R. DEAN RIESS

NUMERICAL ANALYSIS

LEE W. JOHNSON

R. DEAN RIESS

Virginia Polytechnic Institute and State University

NUMERICAL ANALYSIS

ADDISON-WESLEY PUBLISHING COMPANY

Reading, Massachusetts • Menlo Park, California

London • Amsterdam • Don Mills, Ontario • Sydney

Copyright © 1977 by Addison-Wesley Publishing Company, Inc. Philippines copyright 1977 by Addison-Wesley Publishing Company, Inc.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher. Printed in the United States of America. Published simultaneously in Canada. Library of Congress Catalog Card No. 76-14658.

ISBN 0-201-03442-5
BCDEFGHIJK-MA-7987

PREFACE: SUGGESTIONS ON THE USE OF THIS TEXT

In writing this book, our primary goal has been to provide a *textbook* that is well suited for use in an introductory numerical analysis course at the undergraduate level. We view the essential ingredients of such a text to be a proper range of topics, a cohesive and understandable presentation, numerous examples which stress insight into the topics, and large selections of exercises which both reinforce the material of the text and encourage further investigation.

The list of topics is quite extensive, ranging from important classical material to a number of relatively modern concepts. However, the text is by no means an encyclopedia of all possible numerical methods. Rather it presents a selection of commonly used basic procedures together with a comparative analysis of their strengths and weaknesses. Each topic is introduced in its simplest and most understandable form and then developed to a point where the reader has a sound and fundamental background in the subject. With this background, the student may independently delve deeper into the advanced aspects of problems of interest. The examples, for the most part, are kept as simple and direct as possible in order to illustrate a point without obscuring it. There are a few fairly intricate examples, however, which should help the student appreciate the complex nature of typical real-world problems. The exercises include both theoretical and computational problems ranging from the routine to the challenging.

Although this book is intended for use in the classroom, we believe it will serve as a valuable reference book as well since it includes an introduction to several modern topics not usually found in many such texts. These topics include: solutions of over-determined linear systems, spline approximation, the fast Fourier transform, adaptive quadrature, collocation methods for differential equations, and an introduction to some optimization techniques such as quasi-Newton methods, Lagrange multipliers, and linear programming. The text should appeal to engineers, scientists and mathematicians alike, in that it presents computationally efficient and practical methods and also includes sufficient mathematical

theory for a thorough understanding of each method presented. The theoretical material is kept at a minimum and is presented in an expository and intuitive fashion, but is still sufficiently comprehensive so that the reader can understand why each technique works, how efficient it is, and, possibly most important, what can cause it to fail.

The text can be used either for a one-term course in introductory numerical methods or for a full-year numerical analysis course at the junior-senior level. This is possible since the more elementary material is placed at the beginning of each chapter, while the more sophisticated material is near the end of the chapter where it can be omitted without loss of continuity. We have given some illustrations at the end of this preface as to how the text may be adapted to fulfill the purposes of either type of course. There are other alternatives to our suggestions, as the chart of chapter dependencies shows. For example, the instructor of a one-term course may wish to omit the material on eigenvalues entirely and concentrate more on interpolation, quadrature, or differential equations. In either type of course, the instructor can present the material in a different order than it appears in the text. For example, interpolation can be covered first, if so desired. The starred sections are more sophisticated than the others and can be either skimmed or omitted. The depth of coverage of these special sections suggests one way of adjusting the level of the course. The broad range of exercises also provides flexibility in adjusting the level of the course.

The prerequisites for either type of course are basic calculus and a familiarity with the ideas of a matrix and a determinant. The fundamental results from calculus that we use frequently are listed in Chapter 1 and most of the fundamentals of matrix theory are reviewed in Chapter 2. We occasionally introduce material that is not always covered in a freshman/sophomore calculus course, such as norms, inner-products and eigenvalues. In these instances, we give a

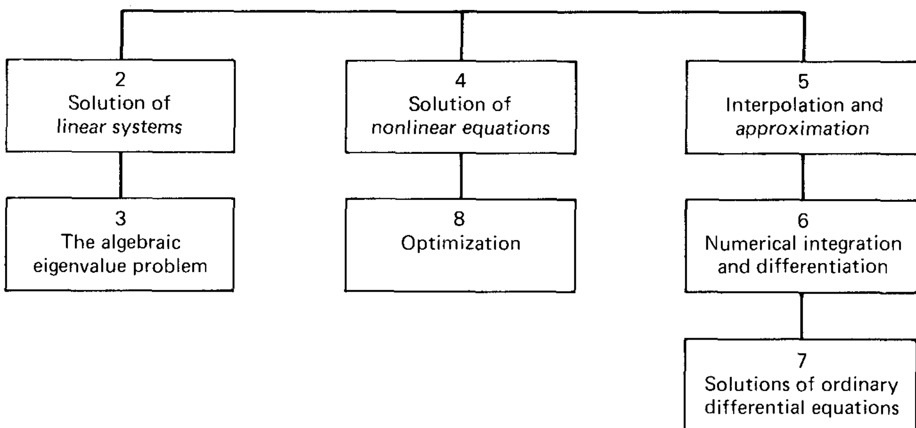

Fig. 1 Relationship of the chapters.

试读结束：需要全本请在线购买： www.ertongbook.com

careful exposition and the reader should have little difficulty understanding the material and its relevance to the particular topic or method being discussed. Whenever possible, material that requires some mathematical maturity is left to the later sections of a chapter or included in the starred (optional) sections.

Although some computational experience can be gained on a desk calculator, it is useful for a student to have a modest programming ability in a language such as FORTRAN. To aid the student in gaining computational experience, we have included a number of fairly simple programs for some of the numerical methods. These programs are in the form of subroutines which are documented and easy to understand since they are written to parallel the statement of the algorithms. In the interest of clarity, we have not tried to include all-purpose, fool-proof codes that handle every possible contingency. Comprehensive, well-tested programs of this sort are available in the literature and in most computer libraries. As we develop techniques to avoid pitfalls that can occur in a particular method the reader should be able to expand the simple programs given in the text to accommodate these techniques.

Chapter 2: Sections 1., 2., 2.1, 2.2, 2.3, 2.4
Chapter 3: Sections 1., 2.
Chapter 4: Sections 1., 2., 3., 3.1, 3.2, 3.3, 3.4, 4., 4.1
Chapter 5: Sections 1., 2., 2.1, 2.2, 2.3, 2.4
Chapter 6: Sections 1., 2., 2.1, 2.2, 2.3, 2.4
Chapter 7: Sections 1., 2., 2.1, 2.2, 2.3, 2.4, 3., 4., 5.

Fig. 2 Suggested list of sections for a self-contained one-term course in numerical methods.

The first chapter presents basic material on rounding errors and floating-point arithmetic. These topics are further discussed and illustrated as they pertain to particular methods in succeeding chapters. However, since a thorough understanding of the effects of rounding and ill-conditioning requires a considerable theoretical background in mathematics and statistics, we primarily present these sources of error intuitively and illustrate them in the methods where they can cause the greatest difficulties.

The chart in Fig. 1 shows the relationships among the chapters of this book. For example, Chapters 5 and 6 are prerequisite to Chapter 7, but Chapter 7 does not depend on the material in Chapters 2, 3 and 4. (In a few instances, some starred sections and starred problems depend on other chapters.) Figure 2 is a list of sections that are appropriate for a comprehensive and self-contained one-term course on numerical methods. The instructor may wish to modify this suggested set of topics deleting some and including others that we have not listed. Furthermore, there are sufficient topics that the instructor of a full-year sequence can selectively omit or skim some, while treating others more carefully.

Blacksburg, Virginia L.W.J
January 1977 R.D.R

CONTENTS

1 Computational and Mathematical Preliminaries

1.1 Introduction 1
1.2 Errors in computations 2
1.3 Rounding errors and floating point arithmetic 3
1.4 Review of fundamental mathematical results 12

2 Solution of Linear Systems of Equations

2.1 Introduction 15
2.2 Direct methods 22
2.2.1 Gauss elimination 22
2.2.2 Operations counts 26
2.2.3 Implementation of Gauss elimination 28
2.2.4 Factorization methods 32
2.3 Error analysis and norms 37
2.3.1 Vector norms 37
2.3.2 Matrix norms 40
2.3.3 Condition numbers and error estimates 43
2.3.4 Iterative improvement 46
2.4 Iterative methods 50
2.4.1 Basic iterative methods 53
2.4.2 Implementation of iterative methods 59
2.5 Least-squares solution of over-determined linear systems 61
*2.6 The Cauchy-Schwarz inequality 65

3 The Algebraic Eigenvalue Problem

3.1 Introduction 68
3.2 The power method 74
3.2.1 Deflation 82

3.2.2 The inverse power method 85
3.2.3 The Rayleigh quotient iteration 88
3.3 Similarity transformations and the characteristic equation 90
3.3.1 Localization of eigenvalues 92
3.3.2 Transformation methods 95
3.3.3 Householder's method 100
*3.4 Schur's theorem and related topics 105

4 Solution of Nonlinear Equations

4.1 Introduction 112
4.2 Bracketing methods 115
4.3 Fixed-point methods 119
4.3.1 The fixed-point problem 120
4.3.2 Rate of convergence of the fixed-point algorithm 125
4.3.3 Newton's method 129
4.3.4 The secant method 134
4.3.5 Newton's method in two variables 138
4.4 Zeros of polynomials 141
4.4.1 Efficient evaluation of a polynomial and its derivatives 143
4.4.2 Bairstow's method 149
4.4.3 Localization of polynomial zeros 153
*4.5 Newton's method in several variables 158

5 Interpolation and Approximation

5.1 Introduction 166
5.2 Polynomial interpolation 171
5.2.1 Efficient evaluation of the interpolating polynomial 173
5.2.2 Interpolation at equally spaced points 178
5.2.3 Error of polynomial interpolation 187
5.2.4 Translating the interval 193
*5.2.5 Polynomial interpolation with derivative data 196
5.2.6 Interpolation by cubic splines 200
5.3 Orthogonal polynomials and least-squares approximations 210
5.3.1 Efficient computation of least-squares approximations 221
*5.3.2 Error estimates for least-squares approximations 226
*5.4 Approximation by rational functions 230

6 Numerical Integration and Differentiation

6.1 Introduction 235
6.2 Interpolatory numerical integration 235
6.2.1 Transforming quadrature formulas to other intervals 239
6.2.2 Newton-Cotes formulas 240
6.2.3 Errors of quadrature formulas 241
6.2.4 Composite rules for numerical integration and the Euler-Maclaurin formula 244

*6.3 Adaptive quadrature 251
6.4 Richardson extrapolation and numerical differentiation 253
6.4.1 Romberg integration 257
6.5 Gaussian quadrature 259
6.5.1 Interpolation at the zeros of orthogonal polynomials 265
6.5.2 Interpolation using Chebyshev polynomials 267
6.5.3 Clenshaw-Curtis quadrature 271
6.6 Trigonometric polynomials and the fast Fourier transform 275
6.6.1 Least-squares fits and interpolation at equally spaced points 277
*6.6.2 The fast Fourier transform 280

7 Numerical Solution of Ordinary Differential Equations

7.1 Introduction 284
7.2 Taylor's series methods 289
7.2.1 Euler's method 290
7.2.2 Taylor's series methods of order k 293
7.2.3 Error analysis for one-step methods 296
7.2.4 Runge-Kutta methods 299
7.3 Predictor-corrector methods 304
7.4 Round-off errors 311
7.5 nth-order differential equations and systems of differential equations 313
7.6 Errors in predictor-corrector methods 316
7.6.1 Step-size control for predictor-corrector methods 320
7.6.2 Stability of predictor-corrector methods 322
7.7 The method of collocation 326
7.8 Boundary value problems for ordinary differential equations 330
7.8.1 Finite difference methods 330
7.8.2 Shooting methods 331
7.8.3 Collocation methods 332

8 Optimization

8.1 Introduction 334
8.2 Extensions of results from calculus 334
8.3 Descent methods 339
8.3.1 Steepest descent 339
8.3.2 Quasi-Newton methods 343
8.4 Lagrange multipliers 346
8.5 Linear programming 352

References 359

Index 363

1

COMPUTATIONAL AND MATHEMATICAL PRELIMINARIES

1.1 INTRODUCTION

In this text we shall concentrate on that portion of numerical analysis that is concerned with the solution of scientific problems utilizing modern high-speed computers. Even from this possibly narrow point of view, numerical analysis is quite widely interdisciplinary. It involves engineering and physics in converting a physical phenomenon into a mathematical model; it involves mathematics in developing techniques for the solution (or approximate solution) of the mathematical equations describing the model; finally it involves computer science for the implementation of these techniques in an optimal fashion for the particular computer available. These three processes are not independent and we should not lose sight of this dependence as we consider a particular aspect of a problem. For example, it is often the case that certain idealistic restrictions must be placed on a physical system in order to obtain a tractable mathematical model. Furthermore, it is possible that mathematical techniques which are derived to "solve" the equations of the model and which serve very well in theory cannot be practically implemented on a particular computer. The task of the numerical analyst is therefore to synthesize these processes and obtain "acceptable" numerical answers. (It is also part of the task to be able to determine when and why "acceptable" answers for a particular problem on a particular computer *cannot* be attained.)

In this text we shall concentrate on the latter two phases of this three-fold process in order to cover more topics, but we will try not to lose sight of the "real world" origins of the problems. In this chapter we shall discuss some basic concepts such as computer representation of numbers, floating-point arithmetic, and rounding errors. We shall also list some basic mathematical results from the calculus which will be useful in the analysis of the numerical methods.

1.2 ERRORS IN COMPUTATIONS

In analyzing the accuracy of numerical results, the numerical analyst should be aware of the possible sources of error in each stage of the computational process and of the extent to which these errors can affect the final answer. We will consider that there are three types of errors which occur in a computation. First, there are errors which we call "initial data" errors. These are errors which arise when the equations of the mathematical model are formed, due to sources such as the idealistic assumptions made to simplify the model, inaccurate measurements of data, miscopying of figures, the inaccurate representation of mathematical constants (for example, if the constant π occurs in an equation, we must replace π by 3.1416 or 3.141593, etc.). Another class of errors, "truncation" errors, occurs when we are forced to use mathematical techniques which give approximate, rather than exact, answers. For example, suppose we use the Maclaurin's series expansion to represent e^x, so that $e^x = 1 + x + x^2/2! + \cdots + x^n/n! + \cdots$. If we want a number that approximates e^β for some β, we must terminate the expansion in order to obtain $e^\beta \approx 1 + \beta + \beta^2/2! + \cdots + \beta^k/k!$. Thus $e^\beta = 1 + \beta + \beta^2/2! + \cdots + \beta^k/k! + E$, where E is the truncation error introduced in the calculation. Truncation errors in numerical analysis usually occur because many numerical methods are iterative in nature, with the approximations theoretically becoming more accurate as we take more iterations. As a practical matter, we must stop the iteration after a finite number of steps, thus introducing a truncation error. The last type of error we shall consider, "round-off" or "rounding" errors, is due to the fact that a computer has a finite word length. Thus most numbers and the results of arithmetic operations on most numbers cannot be represented exactly on a computer. Even though the computer is capable of representing numerical values and performing operations on them, we should be aware of how this is accomplished so that we can understand the error that is produced by inexact representation.

Initial data errors and truncation errors are dependent mostly on the particular problem we are examining, and we shall deal with them as they arise in the context of the different numerical methods we derive throughout the text. The total effect of round-off errors is sometimes dependent on the particular problem, in the sense that the more operations we perform, the more we can probably expect the round-off error to affect the solution. The individual round-off error due to any individual number representation or arithmetic operation is dependent, however, on the particular computer being used, and thus we shall examine the possible sources of this error in the next section, before we introduce any specific numerical methods. We emphasize that it is the effect of total error, from any and all sources, with which we are ultimately concerned. For example, we shall later see problems where a "small" error (no matter from what source) can cause a "large" error in the final solution. Problems of this type are called *ill-conditioned* and must be treated very carefully to obtain an acceptable computed answer.

试读结束：需要全本请在线购买： www.ertongbook.com

There are two ways to measure the size of errors. In analyzing the error of a computation, if we let $\tilde{x}$ represent the "computed solution" to the "true solution" x, then we define the *absolute error* to be $(x - \tilde{x})$ and the *relative error* to be $(x - \tilde{x})/x$. (If the true solution x is zero, then we say the relative error is undefined.) We shall consider these concepts again in later sections, but we briefly pause to mention here that the relative error is usually more significant than the absolute error, and hence we shall try to establish bounds for the relative error whenever possible. To illustrate this point, suppose that in "Computation A" we have $x = 0.5 \times 10^{-4}$ and $\tilde{x} = 0.4 \times 10^{-4}$ while in "Computation B" we have $x = 5000$ and $\tilde{x} = 4950$. The absolute errors are 0.1×10^{-4} and 50, respectively, but the relative errors are 0.2 and 0.01, respectively. Stated differently, Computation A has a 20% error, while Computation B has only a 1% error.

In investigating the effect of the total error in various methods, we shall often mathematically derive an "error bound," which is a limit on how large the error can be. (This applies to both absolute and relative errors.) It is important that the reader realize that the error bound can be much larger than the actual error and that this is often the case in practice. Any mathematically derived error bound must account for the worst possible case that can occur and is often based upon certain simplifying assumptions about the problem which in many particular cases cannot be actually tested. For the error bound to be used in any practical way, the user must have a good understanding of how the error bound was derived in order to know how crude it is, i.e., how likely it is to overestimate the actual error. Of course, whenever possible, our goal is to eliminate or lessen the effects of errors, rather than trying to estimate them after they occur.

1.3 ROUNDING ERRORS AND FLOATING POINT ARITHMETIC

The first type of rounding error that we encounter in performing a computation evolves from the fact that most real numbers cannot be represented exactly on a computer. Superficially, readers are probably not surprised by this statement since they are aware that irrational numbers such as π or e have an infinite non-repeating decimal expansion. Thus they know that even for a hand computation they must use an approximation such as 3.14159 for π or 2.718 for e, and they carry as many digits in their approximations as they feel are necessary for a particular computation. Nevertheless, they realize that once these approximations have been used an error has been introduced into the calculation that can never exactly be corrected.

Since only a finite number of digits can be represented in computer memory, each number x must be represented in some fashion that uses only a fixed number of digits. One of the most common forms is the "floating-point" form, where one position is used to identify the sign of x, a prescribed number of digits are used to represent the "mantissa" or fractional form of x, and an integer is used to

represent the "exponent" or "characteristic" of x with respect to the base b of the representation. (Modern, preferred terminology uses "significand" for mantissa and "exrad" for exponent.) Thus each x can be thought of as being represented by a number $\bar{x}$ of the form $\pm(0.a_1a_2 \cdots a_m) \times (b^c)$ where m is the number of digits allowed in the mantissa, b is the base of the representation, c is the exponent. Additionally, there are two machine-dependent constants, μ and M, such that $\mu \leq c \leq M$. We shall consider three bases: (i) $b = 10$ (decimal)—with which the reader is familiar and which is used on some machines; (ii) $b = 16$ (hexadecimal), which is common to the IBM 360 and 370 series; and (iii) $b = 2$ (binary), which is, in a sense, the most fundamental of the three. To be in proper form we require the mantissa, $a \equiv 0.a_1a_2 \cdots a_m$, to satisfy $|a| < 1 = b^0$ and each a_i, $1 \leq i \leq m$, to be an integer such that $0 \leq a_i \leq b - 1$, with $a_1 \neq 0$ (unless $\bar{x} = 0$). The floating-point representation, $\bar{x}$, is then said to be "normalized."

The floating-point decimal form ($b = 10$) should be familiar to the reader. For example, the decimal number 150.623 is the same as 0.150623×10^3 and can also be regarded as

$$(1 \times 10^2) + (5 \times 10^1) + (0 \times 10^0) + (6 \times 10^{-1}) + (2 \times 10^{-2}) + (3 \times 10^{-3}).$$

Likewise, the binary number 110.011 equals

$$(1 \times 2^2) + (1 \times 2^1) + (0 \times 2^0) + (0 \times 2^{-1}) + (1 \times 2^{-2}) + (1 \times 2^{-3}),$$

and the hexadecimal number 15F.A03 equals

$$(1 \times 16^2) + (5 \times 16^1) + (\mathrm{F} \times 16^0) + (\mathrm{A} \times 16^{-1}) + (0 \times 16^{-2}) + (3 \times 16^{-3}).$$

Note that there are only two digits, 0 and 1, in the binary system, and there are sixteen digits in the hexadecimal system; 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F; where the decimal equivalents of A, B, C, D, E, F are 10, 11, 12, 13, 14, 15; respectively. Also note that the hexadecimal system is a natural extension of the binary system, since $2^4 = 16$, and hence there is precisely one hexadecimal digit for each group of four binary digits ("bits") and vice versa ($0 = (0000)_2$, $7 = (0111)_2$, $\mathrm{A} = (1010)_2$, $\mathrm{F} = (1111)_2$, etc.).

The conversion of an integer from one system to another is fairly simple and can probably best be presented in terms of an example. Let $k = 275$ in decimal form, that is, $k = (2 \times 10^2) + (7 \times 10^1) + (5 \times 10^0)$. Now $(k/16^2) > 1$, but $(k/16^3) < 1$, so in hexadecimal form k can be written as $k = (\alpha_2 \times 16^2) + (\alpha_1 \times 16^1) + (\alpha_0 \times 16^0)$. Now, $275 = 1(16^2) + 19 = 1(16^2) + 1(16) + 3$, and so the decimal integer, 275, can be written in hexadecimal form as 113, that is, $(275)_{10} = (113)_{16}$. The reverse process is even simpler. For example, $(5\mathrm{C}3)_{16} = 5(16^2) + 12(16) + 3 = 1280 + 192 + 3 = (1475)_{10}$. Conversion of a hexadecimal fraction to a decimal is similar. For example, $(0.2\mathrm{A}8)_{16} = (2/16) + (\mathrm{A}/16^2) + (8/16^3) = (2(16^2) + 10(16) + 8)/16^3 = (680)/4096 = (0.166)_{10}$, (carrying only three digits in the decimal form). Conversion of a decimal fraction to hexadecimal (or

binary) proceeds as in the following example. Consider the number $r_1 = 1/10 = 0.1$ (decimal form). Then there exist constants $\{\alpha_k\}_{k=1}^{\infty}$ such that

$$r_1 = 0.1 = \alpha_1/16 + \alpha_2/16^2 + \alpha_3/16^3 + \alpha_4/16^4 + \cdots.$$

Now, $16r_1 = 1.6 = \alpha_1 + \alpha_2/16 + \alpha_3/16^2 + \alpha_4/16^3 + \cdots$. Thus $\alpha_1 = 1$ and $r_2 \equiv 0.6 = \alpha_2/16 + \alpha_3/16^2 + \alpha_4/16^3 + \cdots$. Again, $16r_2 = 9.6 = \alpha_2 + \alpha_3/16 + \alpha_4/16^2 + \cdots$, so $\alpha_2 = 9$ and $r_3 \equiv 0.6 = \alpha_3/16 + \alpha_4/16^2 + \cdots$. From this stage on we see that the process will repeat itself, and so we have $(0.1)_{10}$ equals the infinitely repeating hexadecimal fraction, $(0.1999\cdots)_{16}$. Since $1 = (0001)_2$ and $9 = (1001)_2$ we also have the infinite binary expansion

$$r_1 = (0.1)_{10} = (0.1999\cdots)_{16} = (0.0001\ 1001\ 1001\ 1001\cdots)_2.$$

From the above example we begin to discern one problem of number representation. Not only do we have problems with irrational numbers and infinite repeating decimal expansions such as $1/3 = 0.333\cdots$, but we also see that an m-digit terminating fraction with respect to one base may not have an n-digit terminating representation in another base. (If we were performing an iteration on a hexadecimal machine, the above example suggests that we should probably choose a step-size of $h = 1/16$ over $h = 1/10$, $h = 1/1024 = 1/2^{10}$ over $h = 1/1000$, etc., if at all possible.) In Table 1.1 we have taken several integers k, formed the reciprocals, $1/k$, and added the reciprocal to itself k times. The theoretical result of each computation should, of course, equal 1. The calculations were performed on a six-digit hexadecimal machine.[†]

Table 1.1

k	Sum($1/k$)	k	Sum($1/k$)	k	Sum($1/k$)
2	0.1000000E 01	9	0.9999999E 00	16	0.1000000E 01
3	0.9999999E 00	10	0.9999996E 00	⋮	
4	0.1000000E 01	11	0.9999997E 00	1000	0.9999878E 00
5	0.9999999E 00	12	0.9999998E 00	1006	0.9999912E 00
6	0.9999998E 00	13	0.9999999E 00	1012	0.9999843E 00
7	0.9999999E 00	14	0.9999995E 00	1018	0.9999678E 00
8	0.1000000E 01	15	0.9999999E 00	1024	0.1000000E 01

We must, of course, realize that part of the error in the above table is due to the round-off from addition. We shall discuss this momentarily, but for now the reader should notice the relative accuracies for the values of k which were powers of 2. For example, compare $k = 1000$ to $k = 1024 = 2^{10}$.

We next consider how numbers are represented in an m-digit machine, particularly those numbers for which m digits are not sufficient to represent the

[†] Most of the computer programs in this text were run on an IBM 370/158.

number exactly. For example, how might a number like $\frac{1}{3}$ be represented on a 5-digit decimal computer, since $\frac{1}{3} = 0.3333333\cdots$. As a general example, suppose a real number x is given exactly by

$$x = \pm(0.\tilde{a}_1\tilde{a}_2\cdots\tilde{a}_m\tilde{a}_{m+1}\cdots)\times 10^c, \qquad \tilde{a}_1 \neq 0$$

and suppose we want to represent x in an m-digit decimal computer. There are two common ways of representing x. First, we can simply let $a_k = \tilde{a}_k$, $1 \leq k \leq m$, discard the remaining digits, and let the computer representation be

$$\bar{x} = \pm(0.a_1a_2\cdots a_m)\times 10^c.$$

This is known as "chopping." The other representation is the familiar "symmetric rounding" process which is equivalent to adding $5 \times 10^{c-m-1}$ to x and then chopping (we think of adding 5 to $\tilde{a}_{m+1}$). Of course, if $c < \mu$ or $c > M$, the number lies outside the range of admissable computer representations. If $c < \mu$, (underflow), it is common to regard x as zero, notify the user, and continue further computation. If $c > M$, then overflow results. It is usually deemed not worth the added expense to use symmetric rounding and most machines simply chop. Whether a representation is obtained via chopping or symmetric rounding, we shall hereafter refer to the machine representation, $\bar{x}$, as being "rounded." Note that the above was illustrated in decimal form for simplicity; the analogy can be carried over to other bases. For example, if x has the hexadecimal form

$$x = \pm(0.\tilde{a}_1\tilde{a}_2\cdots\tilde{a}_m\tilde{a}_{m+1}\cdots)\times 16^c, \qquad \tilde{a}_1 \neq 0,$$

then the "decimal" point is really a "hexadecimal" point, and

$$x = \left(\sum_{k=1}^{\infty}(\tilde{a}_k \times 16^{-k})\right)\times 16^c.$$

The chopped form is still obtained by deleting $\tilde{a}_k$, $k \geq m+1$. Symmetric rounding is done by adding $8 \times 16^{c-m-1}$ to x and then chopping.

Returning to decimal form for simplicity, we first consider the error in symmetric rounding. If $\tilde{a}_{m+1} \geq 5$, then

$$x - \bar{x} = (\pm 0.\tilde{a}_1\cdots\tilde{a}_m\tilde{a}_{m+1}\cdots)\times 10^c - [(\pm 0.\tilde{a}_1\cdots\tilde{a}_m\tilde{a}_{m+1}) + (\pm 0.0\cdots 05)]\times 10^c.$$

If $\tilde{a}_{m+1} < 5$, then

$$x - \bar{x} = (\pm 0.\tilde{a}_1\cdots\tilde{a}_m\tilde{a}_{m+1}\cdots)\times 10^c - (\pm 0.\tilde{a}_1\cdots\tilde{a}_m)\times 10^c.$$

In either case we have that $|x - \bar{x}| < 0.5 \times 10^{c-m}$, and this is a bound on the absolute error. To get a bound on the relative error, we note that since $\tilde{a}_1 \neq 0$, then $|x| \geq 0.1 \times 10^c$. Thus the relative error satisfies

$$\frac{|x - \bar{x}|}{|x|} \leq \frac{0.5 \times 10^{c-m}}{0.1 \times 10^c} = 5 \times 10^{-m} = 0.5 \times 10^{-m+1}.$$

In a similar manner we can show that the relative error from chopping is $1 \times 10^{-m+1}$. It is interesting to note that neither of these relative error bounds depend on the magnitude of x, that is, the size of c. Rather they depend only on the value of m, which is thus said to be the number of "significant digits" of the computer. For example, the IBM System 360 and 370 are hexadecimal with a mantissa of $m = 6$. The relative error from chopping in number representation thus does not exceed 1×16^{-5}. To compare this with the accuracy of relative errors we expect in the decimal mode, we set $1 \times 16^{-5} = 1 \times 10^{-m+1}$. Solving for m yields $m \approx 7$. Thus we have approximately seven-significant-digit decimal accuracy with respect to the relative error of the representation. This does *not* mean that any seven-digit decimal number can be represented exactly on this machine as the previous example of $x = 1/10$ shows. It does say, however, that $|x - \bar{x}|/|x| \leq 10^{-6}$ (approximately, since $m \approx 7$).

In the discussion above, we analyzed the error made by replacing the true value of x by its machine representation $\bar{x}$. Now we wish to assess the effect of each arithmetic operation $(+, -, \cdot, \div)$ to see how errors propagate. Let us use $\bar{x}$ to denote the machine approximation to the true value x, where the error, $e(x) = x - \bar{x}$, includes *all* errors that have been made in going from x to $\bar{x}$, that is, $e(x)$ not only includes the error of representation but also includes errors from previous calculations, initial data errors, etc. In other words, $e(x)$ includes all errors from the beginning of the calculation that have led to any discrepancies between x and $\bar{x}$. With $e(y)$ defined similarly for any quantity y, we investigate the error resulting from the "machine addition" of x and y. Now,

$$x + y = (\bar{x} + e(x)) + (\bar{y} + e(y)) = (\bar{x} + \bar{y}) + (e(x) + e(y)).$$

At first glance it seems that the error of the addition is merely the sum of the individual errors. However, this is not always true, since even though $\bar{x}$ and $\bar{y}$ can be represented exactly on the machine, it is *not* necessarily true that their sum can also be, i.e., it does not follow that $\bar{x} + \bar{y} = \overline{\bar{x} + \bar{y}}$. For example, consider a four-digit floating-point machine and let $\bar{x} = 0.9621 \times 10^0$ and $\bar{y} = 0.6732 \times 10^0$. Then $\bar{x} + \bar{y} = 1.6353 \times 10^0$. Now it is not uncommon for an m-digit machine to perform arithmetic operations in a $2m$-digit accumulator and then to round the answer. Assuming this to be true, we have $\overline{\bar{x} + \bar{y}} = 0.1635 \times 10^1$. Returning to our general analysis we see that the true error, $z - \bar{z}$, where $z = x + y$, is actually $(e(x) + e(y))$ *plus* the error between $\bar{x} + \bar{y}$ and $\overline{\bar{x} + \bar{y}}$.

Before examining the other three arithmetic operations, let us consider addition somewhat further. We can see immediately that addition can lead to overflow, for instance, $\bar{x} = 0.9621 \times 10^M$ and $\bar{y} = 0.6732 \times 10^M$, thus $\bar{x} + \bar{y}$ does not fall within the range of the computer. Another factor that we must consider is that before a machine can perform an addition, it must align the decimal points. For example, again let $m = 4$ with $\bar{x}_1 = 0.5055 \times 10^4$ and $\bar{x}_2 = \bar{x}_3 = \cdots \bar{x}_{11} = 0.4000 \times 10^0$. To perform the addition, $\bar{x}_1 + \bar{x}_2$, the computer must shift the decimal four places to the left in $\bar{x}_2$ and form $\bar{x}_1 + \bar{x}_2 = (0.5055 \times 10^4) +$

$(0.00004 \times 10^4) = (0.50554 \times 10^4)$, which rounds to $\overline{\bar{x}_1 + \bar{x}_2} = 0.5055 \times 10^4 = \bar{x}_1$. Continuing, we see that

$$\overline{(\cdots(((\bar{x}_1 + \bar{x}_2) + \bar{x}_3) + \bar{x}_4) + \cdots + \bar{x}_{11})} = \bar{x}_1,$$

but

$$\overline{(\cdots(((\bar{x}_{11} + \bar{x}_{10}) + \bar{x}_9) + \bar{x}_8) + \cdots + \bar{x}_1)} = 0.5059 \times 10^4$$

which is the correct answer. Thus the machine calculates $\sum_{i=0}^{10} \bar{x}_{11-i}$ correctly, but not the sum $\sum_{i=1}^{11} \bar{x}_i$. This example illustrates the rule of thumb that if we have several numbers of the same sign to add, we should add them in ascending order of magnitude to minimize the propagation of round-off error. The mathematical foundation underlying this statement is the fact that machine addition is not associative, i.e., it can happen that

$$\overline{(\bar{x} + \bar{y}) + \bar{z}} \neq \overline{\bar{x} + (\bar{y} + \bar{z})}.$$

The following numerical examples were run to illustrate this phenomenon. (We used a hexadecimal machine with $m = 6$.) Letting $x = 1048576 = 16^5 = \bar{x}$ and $y = z = 1/2 = 8/16 = \bar{y} = \bar{z}$, our machine results were

$$(\bar{x} + \bar{y}) + \bar{z} = 0.1048576\text{E}07$$

and

$$\bar{x} + (\bar{y} + \bar{z}) = 0.1048577\text{E}07,$$

as our above analysis leads us to expect. Letting $w_0 = 1048576 = 16^5 = \overline{w_0}$ and $w_k = 1/16 = \bar{w}_k$, $1 \le k \le 256$, we also obtained the following results (adding in the order indicated by the sum):

$$\sum_{k=0}^{256} w_k = 0.1048576\text{E}07 = w_0$$

but

$$\sum_{k=0}^{256} w_{256-k} = 0.1048592\text{E}07,$$

the latter being the correct result (which we can easily check by hand). As a final example we computed the sum, $S \equiv \sum_{k=1}^{999} (1/k(k+1)) \equiv 0.999$, by adding forwards and backwards and obtained

$$S \approx 0.9989709\text{E}00 \text{ (forwards)}, \qquad S \approx 0.9989992\text{E}00 \text{ (backwards)}.$$

The error analysis for subtraction is much the same as addition, in that

$$x - y = (\bar{x} - \bar{y}) + (e(x) - e(y)),$$

but now we have the problem that subtraction can cause loss of significant digits.

试读结束：需要全本请在线购买： www.ertongbook.com