R. W. HAMMING

Numerical Methods for Scientists and Engineers

SECOND EDITION

McGRAW-HILL
BOOK COMPANY
New York
St. Louis
San Francisco
Düsseldorf
Johannesburg
Kuala Lumpur
London
Mexico
Montreal
New Delhi
Panama
Rio de Janeiro
Singapore
Sydney
Toronto

R. W. HAMMING
Bell Telephone Laboratories

Numerical Methods for Scientists and Engineers

SECOND EDITION

This book was set in Times New Roman.
The editors were Jack L. Farnsworth and J. W. Maisel;
the designer was Barbara Ellwood;
and the production supervisor was Ted Agrillo.
The drawings were done by Eric G. Hieber.
The printer and binder was R. R. Donnelley & Sons Company.

Library of Congress Cataloging in Publication Data

Hamming, Richard Wesley, 1915–

Numerical methods for scientists and engineers.

(Series in pure & applied math)
Bibliography: p.
1. Electronic data-processing—Numerical analysis.
I. Title.
QA297.H28 1973 519.4 72-12643
ISBN 0-07-025887-2

NUMERICAL METHODS
FOR SCIENTISTS
AND ENGINEERS

1234567890 DODO79876543

PREFACE

There has been much progress in the 10 years since the first edition was written, but of the many books that have appeared on the topic none has put the emphasis on the frequency approach and its use in the solution of problems. For these reasons, a second edition seems necessary.

The material has been extensively rearranged, rewritten, and added to, so that in some respects it is a new book; however, the main aims, style, and motto have not changed.

As always, the author is greatly indebted to others for much that is in the book. Most important are his management and colleagues at the Bell Telephone Laboratories. Professor Roger Pinkham has over the years been a constant source of stimulation and inspiration. It would take a list of at least 100 names to thank all who have contributed to some extent, and at the top of this list would be M. P. Epstein. My thanks also go to all the unmentioned people on the list and to A. Ralson for many helpful suggestions. Thanks also to Mrs. Jeannie Waddel for typing and helping to organize the manuscript.

R. W. HAMMING

CONTENTS

* Starred sections may be omitted.

II Polynomial Approximation—Classical Theory

III Fourier Approximation—Modern Theory

IV Exponential Approximation

V Miscellaneous

* Starred sections may be omitted.

PART I

Fundamentals and Algorithms

1

AN ESSAY ON NUMERICAL METHODS

1.1 THE FIVE MAIN IDEAS

Numerical methods use numbers to simulate mathematical processes, which in turn usually simulate real-world situations. This implies that there is a *purpose* behind the computing. To cite the motto of the book, The Purpose of Computing Is Insight, Not Numbers. This motto is often thought to mean that the numbers from a computing machine should be read and used, but there is much more to the motto. The choice of the particular formula, or algorithm, influences not only the computing but also how we are to understand the results when they are obtained. The way the computing progresses, the number of iterations it requires, or the spacing used by a formula, often sheds light on the problem. Finally, the same computation can be viewed as coming from different models, and these different views often shed further light on the problem. *Thus computing is, or at least should be, intimately bound up with both the source of the problem and the use that is going to be made of the answers—it is not a step to be taken in isolation from reality.*

Much of the knowledge necessary to meet this goal comes from the field of application and therefore lies outside a general treatment of numerical methods. About all that can be done is to supply a rich assortment of methods and to

comment on their relevance in general situations. This art of connecting the specific problem with the computing is important, but it is best taught in connection with a field of application.

The second main idea is a consequence of the first. If the purpose of computing is insight, not numbers, as the motto states, then it is necessary to *study families and to relate one family to another when possible*, and to avoid isolated formulas and isolated algorithms. In this way a sensible choice can be made among the alternate ways of doing the problem, and once the computation is done, alternate ways of viewing the results can be developed. Thus, hopefully, the insight can arise. For these reasons we tend to concentrate on systematic methods for finding formulas and avoid the isolated, cute result. It is somewhat more difficult to systematize algorithms, but a unifying principle has been found.

This is perhaps the place to discuss some of the differences between numerical methods and numerical analysis (as judged by the corresponding textbooks). Numerical analysis seems to be the study in depth of a few, somewhat arbitrarily selected, topics and is carried out in a formal mathematical way devoid of relevance to the real world. Numerical methods, on the other hand, try to meet the need for methods to cope with the potentially infinite variety of problems that can arise in practice. The methods given are generally chosen for their wide applicability in creating formulas and algorithms as well as for the particular result being found at that point.

The third major idea is *roundoff error*. This effect arises from the finite nature of the computing machine which can only deal with finitely represented numbers. But the machine is used to simulate the mathematician's number system which uses infinitely long representations. In the machine the fraction $\frac{1}{3}$ becomes the terminated decimal 0.333...3 with the obvious roundoff effect. At first this approximation does not seem to be very severe since usually a minimum of eight decimal places are carried at every step, but the incredible number of arithmetic operations that can occur in a problem lasting only a few seconds is the reason that roundoff plays an important role. *The greatest loss of significance in the numbers occurs when two numbers of about the same size are subtracted so that most of the leading digits cancel out, and unless care is taken in advance, this can happen almost any place in a long computation.*

Most books on computing stress the estimation of roundoff, especially the bounding of roundoff, but we shall concentrate *on the avoidance of roundoff*. It seems better to avoid roundoff than to estimate what did not have to occur if common sense and a few simple rules had been followed *before* the problem was put on the machine.

The fourth main idea is again connected with the finite nature of the machine, namely that many of the processes of mathematics, such as differentiation and in-

tegration, imply the use of a limit which is an infinite process. The machine has finite speed and can only do a finite number of operations in a finite length of time. This effect gives rise to the *truncation error* of a process.

We shall generally first give an exact expression for the truncation error and deduce from it various bounds. A moment's thought should reveal that if we had an exact expression, then it would be practically useless because to know the exact error is to know the exact answer. However, the exact-error expression is very useful in studying families of formulas, and it provides a starting point for a variety of error bounds.

The fifth main idea is *feedback*, which means, as its name implies, that numbers produced at one stage are fed back into the computer to be processed again and again; the program has a loop which uses the output of one cycle as the input for the next cycle. This feedback situation is very common in computing, as it is a very powerful tool for solving many problems.

Feedback leads immediately to the associated idea of *stability* of the feedback loop—will a small error grow or decay through the successive iterations? The answer may be given loosely in two equivalent ways: first, if the feedback of the error is too strong and is in the direction to eliminate the error (technically, negative feedback), then the system will break into an oscillation that grows with time; second and equivalently, if the feedback is delayed too long, the same thing will happen.

A simple example that illustrates feedback instability is the common home shower. Typically the shower begins with the water being too cold, and the user turns up the hot water to get the temperature he wants. If the adjustment is too strong (he turns the knob too far), he will soon find that the shower is too hot, whereupon he rapidly turns back to cold and soon finds it is too cold. If the reactions are too strong, or alternately the total system (pipes, valve, and human) is too slow, there will result a "hunting" that grows more and more violent as time goes on. Another familiar example is the beginning automobile driver who overreacts while steering and swings from side to side of the street. This same kind of behavior can happen for the same reasons in feedback computing situations, and therefore the stability of a feedback system needs to be studied before it is put on the computer.

1.2 SECOND-LEVEL IDEAS

Below the main ideas in Sec. 1.1 are about 50 second-level ideas which are involved in both theoretical and practical work. Some of these are now discussed.

At the foundation of all numerical computing are the actual numbers

themselves. The floating-point number system used in most scientific and engineering computations is significantly different from the mathematician's usual number system. The floating-point numbers are not equally spaced, and the numbers do not occur with equal frequency. For example, it is well known that a table of physical constants will have about 60 percent of the numbers with a leading digit of 1, 2, or 3, and the other digits—4, 5, 6, 7, 8, and 9—comprise only 40 percent.

Although this number system lies at the foundation of most of computing, it is rarely investigated with any care. People tend to start computing, and only after having frequent trouble do they begin to look at the system that is causing it.

Immediately above the number system is the apparently simple matter of evaluating functions accurately. Again people tend to think that they know how to do it, and it takes a lot of painful experience to teach them to examine the processes they use before putting them on a computer.

These two mundane, pedestrian topics need to be examined with the care they deserve *before* going on to more advanced matters; otherwise they will continually intrude in later developments.

Perhaps the simplest problem in computing is that of finding the zeros of a function. In the evaluation of a function near a zero there is almost exact cancellation of the positive and negative parts, and the two topics we just discussed, roundoff of the numbers and function evaluation, are basic, since if we do not compute the function accurately, there can be little meaning to the zeros we find. Because of the discrete structure of the computer's number system it is very unlikely that there will be a number x which will make the function $y = f(x)$ exactly zero. Instead, we generally find a small interval in which the function changes sign. The size of the interval we can use is related to the size of the argument x, since for large x the number system has a coarse spacing and for x small (in size) it has a fine spacing. This is one of the reasons that the idea of the *relative error*

$$\text{Relative error} = \left| \frac{\text{true} - \text{calculated}}{\text{true}} \right|$$

plays such a leading role in scientific and engineering computations. Classical mathematics uses the *absolute error*

$$\text{Absolute error} = |\text{true} - \text{calculated}|$$

most of the time, and it requires a positive effort to unlearn the habits acquired in the conventional mathematics courses. The relative error has trouble near places where the true value is approximately zero, and in such cases it is customary to use as the denominator

$$\max\{|x|, |f(x)|\}$$

where $f(x)$ is the function computed at x.

The problem of finding the complex zeros of an analytic function occurs so often in practice that it cannot be ignored in a course on numerical methods, though it is almost never mentioned in numerical analysis. A simple method resembling one used to find the real zeros is very effective in practice.

In the special case of finding all the zeros of a polynomial the fact that the number of zeros (as well as other special characteristics) is known in advance makes the problem easier than for the general analytic function. One of the best methods for finding them is an adaptation of the usual Newton's method for finding real zeros, and this discussion is used to extend, as well as to analyse further, Newton's method. It is only in situations in which a careful analysis can be made that Newton's method is useful in practice; otherwise its well-known defects outweigh its virtues.

What makes the problem of finding the zeros of a polynomial especially important, besides its frequency, is the use made of the zeros found. The method is a good example of the difference between the mathematical approach and the engineering approach. The first merely tries to find some numbers which make the function close to zero, while the second recognizes that a pair of "close" zeros will give rise to severe roundoff troubles when used at a later stage. In isolation the problem of finding the zeros is not a realistic problem since the zeros are to be used, not merely admired in a vacuum. Thus what is wanted in most practice is the finding of the multiple zeros as multiple zeros, not as close, separate ones. Similarly, zeros which are purely imaginary are to be preferred to ones with a small real part and a large imaginary part, *provided* the difference can reasonably be attributed to uncertainties in the underlying model.

Another standard algorithmic problem both in mathematics and in the use of computation to solve problems is the solution of simultaneous linear equations. Unfortunately much of what is commonly taught is usually not relevant to the problem as it occurs in practice; nor is any completely satisfactory method of solution known at present. Because the solution of simultaneous linear equations is so often a standard library package supplied by the computing center and because the corresponding description is so often misleading, it is necessary to discuss the limitations (and often the plain foolishness) of the method used by the package. Thus it is necessary to examine carefully the obvious flaws and limitations, rather than pretending they do not exist.

The various algorithms for finding zeros, solving simultaneous linear equations, and inverting matrices are the classic algorithms of numerical analysis. Each is usually developed as a special trick, with no effort to show any underlying principles. The idea of an *invariant algorithm* provides one common idea linking, or excluding, various methods. An invariant algorithm is one that in a very real sense attacks the problem rather than the particular representation supplied to the

computer. The idea of an invariant algorithm is actually fairly simple and obvious once understood. In many kinds of problems there are one or more classes of transformations that will transform one representation of the equations into another of the same form. For example, given a polynomial

$$P(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_0 = 0$$

the transformation of multiplying the equation by any nonzero constant does not really change the problem. Similarly, when $a_0 \neq 0$, replacing x by $1/x$ while also multiplying the equation by x^n merely reverses coefficients. These transformations form a group (provided we recognize the finite limitations of computing), and it is natural to ask for algorithms that are invariant with respect to this group, where invariant means that if the problem is transformed to some equivalent form, then the algorithm uses, at all stages, the equivalent numbers (within roundoff, of course). In a sense the invariance is like dimensional analysis—the scaling of the problem should scale the algorithm in exactly the same way. It is more than dimensional analysis since, as in the example of the polynomial, some of the transformations to be used in the problem may involve more than simple scaling.

It is surprising how many common algorithms do not satisfy this criterion. The principle does more than merely reject some methods; it also, like dimensional analysis, points the way to proper ones by indicating possible forms that might be tried.

1.3 THE FINITE DIFFERENCE CALCULUS

After examining the simpler algorithms, it is necessary to develop more general tools if we are to go further. The *finite difference calculus* provides both the notation and the framework of ideas for many computations. The finite difference calculus is analogous to the usual infinitesimal calculus. There are the difference calculus, the summation calculus, and difference equations. Each has slight variations from the corresponding infinitesimal calculus because instead of going to the limit, the finite calculus stops at a fixed step size. This reveals why the finite calculus is relevant to many applications of computing: in a sense it *undoes* the limiting process of the usual calculus. It should be evident that if a limit process cannot be undone, then there is a very real question as to the soundness of the original derivation, because it is usually based on constructing a believable finite approximation and then going to the limit.

The finite difference calculus provides a tool for estimating the roundoff effects that appear in a table of numbers *regardless* of how the table was computed. This tool is of broad and useful application because instead of carefully studying

each particular computation, we can apply this general method without regard to the details of the computation. Of course, such a general method is not as powerful as special methods hand-tailored to the problem, but for much of computation it saves both trouble and time.

The summation calculus provides a natural tool for approaching the very common (and often neglected) problem of the summation of infinite series, which is the simplest of the limiting processes (since the index n of the number of terms taken runs through the integers only).

The solution of finite difference equations is analogous to the solution of differential equations, especially the very common case of linear difference equations with constant coefficients, which is a valuable tool for the study of feedback loops and their stability. Thus finite difference equations have both a practical and a theoretical value in computing.

1.4 ON FINDING FORMULAS

Once past the easier algorithms and tools for doing simple things in computing, it is natural to attack one of the central problems of numerical methods, namely, the approximation of infinite operations (operators) by finite methods. Interpolation is the simplest case. In interpolation we are given some samples of the function, say, $y(-1)$, $y(0)$, and $y(1)$, and we are asked to *guess* at the missing values—to read between the lines of a table. While it is true that because of the finite nature of the number system used there are only a finite number of values to be found, nevertheless this number is so high that it might as well be infinite. Thus interpolation is an infinite operator to be approximated.

There is no sense to the question of interpolation unless some additional assumptions are made. The *classical assumption* is that given $n + 1$ samples of the function, these samples determine a unique polynomial of degree n, and this polynomial is to be used to give the interpolated values. With the above data consisting of three points, the quadratic through these points is

$$P(x) = \frac{x(x-1)}{2}y(-1) + (1-x^2)y(0) + \frac{x(x+1)}{2}y(1)$$

We are to use this polynomial $P(x)$ as if it were the function. This method is known as the *exact matching* of the function to the data.

The error of this interpolation can be expressed as the $(n + 1)$st derivative (of the original function) evaluated at some generally unknown point θ in the interval. Unfortunately in practice it is rare to have any idea of the size of this derivative.

For samples of the function we may use not only function values $y(x)$ but also values of the derivatives $y'(x)$, $y''(x)$, etc., at various points. For example, the cubic exactly matching the data $y(0)$, $y(1)$, $y'(0)$, and $y'(1)$ is

$$P(x) = (1 - 3x^2 + 2x^3)y(0) + (3x^2 - 2x^3)y(1) + (x - 2x^2 + x^3)y'(0) + (x^3 - x^2)y'(1)$$

It is important to use analytically found derivatives when possible. Then we can usually get a higher order of approximation at little extra cost since generally once the function values are computed, the derivatives are relatively easy to compute. No new radicals, logs, exponentials, etc., arise, and these are the time-consuming parts of most function evaluation. Of course a sine goes into a cosine when differentiated, but this is about the only new term needed for the higher derivatives. Even the higher transcendental functions, like the Bessel functions, satisfy a second-order linear differential equation, and once both the function and the first derivative are found, the higher derivatives can be computed from the differential equation and its derivatives (which are easy to compute). Thus we shall emphasize the use of derivatives as well as function values for our samples.

Although a wide variety of function and derivative values may be used to determine the interpolating polynomial, there are some sets, rather naturally occurring, for which $n + 1$ data samples do not determine an nth-degree polynomial. Perhaps the best example is the data $y(-1)$, $y(0)$, $y(1)$, $y''(-1)$, $y''(0)$, and $y''(1)$ which do not determine a fifth-degree polynomial—the positions and accelerations at three equally spaced points do not determine a quintic in general.

The classic method for finding formulas for other infinite operators, such as integration and differentiation, is to use the interpolating polynomial as if it were the function and then to apply the infinite operator to the polynomial. For example, if we wish to find the integral of a function from -1 to $+1$, given the values $y(-1)$, $y(0)$, and $y(1)$, we find the interpolating quadratic as above and integrate it to get the classical Simpson's formula:

$$\int_{-1}^{1} y(x)\,dx = \tfrac{1}{3}y(-1) + \tfrac{4}{3}y(0) + \tfrac{1}{3}y(1)$$

This process is called *analytic substitution*; in place of the function we could not handle we take some samples, exactly match a polynomial to the data, and finally analytically operate on this polynomial. This is the classical method for finding formulas. It is a two-step method: find the interpolating function and then apply the operator to this function.

There is another direct method that is *almost* equivalent to the analytic-substitution method. In this method we make the formula true for a sequence of