CS 147

PITFALLS IN COMPUTATION, OR WHY A MATH BOOK ISN'T ENOUGH

BY

GEORGE E. FORSYTHE

TECHNICAL REPORT NO. CS 147

JANUARY 1970

COMPUTER SCIENCE DEPARTMENT

School of Humanities and Sciences

STANFORD UNIVERSITY

44

PITFALLS IN COMPUTATION, OR WHY A MATH BOOK ISN'T ENOUGH

George E. Forsythe *

1. Introduction

Why do students take mathematics in college and university? I see two reasons: (i) To learn the structure of mathematics itself, because they find it interesting. (ii) To apply mathematics to the solution of problems they expect to encounter in their own fields, whether it be engineering, physics, economics, or whatever.

I am sure that (ii) motivates far more students than (i). Moreover, most solutions of major mathematical problems involve the use of automatic digital computers. Hence we may justifiably ask what mathematics courses have to say about carrying out mathematical work on a computer. This question motivates my paper.

I am not in a mathematics department, and tend to moralize about them. If the reader prefers not to be lectured to, I invite him to ignore the preaching and just pay attention to the numerical phenomena for their own sake.

I want to acknowledge the help of Mr. Michael Malcolm in critizing the manuscript and doing the computations with a special floating decimal arithmetic simulator he wrote for Stanford's hexadecimal computer.

*The preparation of this manuscript was supported in part by the Office of Naval Research (NR 044 211), the National Science Foundation (GJ 798), and the Atomic Energy Commission (Stanford PA #18). This material was presented by invitation to the Mathematical Association of America in Eugene, Oregon, 25 August 1969.

2. Nature of computers

An automatic digital computer is a general-purpose machine. The bits of information in its store can be used to represent any quantifiable objects -- e.g., musical notes, letters of the alphabet, elements of a finite field, integers, rational numbers, parts of a graph, etc. Thus such a machine is a general abstract tool, and the generality of computing makes computer science an important topic, just as mathematics and natural language are important.

In the use of computers to represent letters of the alphabet, elements of a finite field, integers, etc., there is no error in the representation, nor in the processes that operate upon the quantities so represented. The problems in dealing with integers (to select one example) on computers are of the following types: Is there enough storage to contain all the integers I need to deal with? Do I know a process that is certain to accomplish my goal on the integers stored in the computer? Have I removed the logical errors ("bugs") from my computer representation of this process? Is this the fastest possible process or, if not, does it operate quickly enough for me to get (and pay for) the answers I want?

The above problems are not trivial; there are surely pitfalls in dealing with them; and it is questionable whether math books suffice for their treatment. But they are not the subject of this paper. This paper is concerned with the simulated solution on a digital computer of the problems of algebra and analysis dealing with real and complex numbers. Such problems occur everywhere in technology -- for example, whenever it is required to solve a differential equation or a system of algebraic equations.

There are four properties of computers that are relevant to their use in the numerical solution of problems of algebra and analysis, and are causes of many pitfalls:

i) Computers use not the real number system, but instead a simulation of it called a "floating-point number system." This introduces the problem of <u>round-off</u>.

ii) The speed of computer processing permits the solution of very large problems. And frequently (but not always) large problems have answers that are much more <u>sensitive</u> to perturbations of the data than small problems are.

iii) The speed of computer processing permits many more operations to be carried out for a reasonable price than were possible in the pre-computer era. As a result, the <u>instability</u> of many processes is conspicuously revealed.

iv) Normally the intermediate results of a computer computation are hidden in the store of the machine, and never known to the programmer. Consequently the programmer must be able to detect errors in his process without seeing the warning signals of possible error that occur in desk computation, where all intermediate results are in front of the problem solver. Or, conversely, he must be able to prove that his process cannot fail in any way.

3. Floating-point number system

The badly named real number system is one of the triumphs of the human mind. It underlies the calculus and higher analysis to such a degree that we may forget how impossible it is to deal with real numbers in the real world of finite computers. But, however much the real number system simplifies analysis, practical computing must do without it.

Of all the possible ways of simulating real numbers on computers, one is most widely used today -- the floating-point number systems. Here a number base β is selected, usually 2, 8, 10, or 16. A certain integer s is selected as the number of significant digits (to base β) in a computer number. An integer exponent e is associated with each nonzero computer number, and e must lie in a fixed range, say

$$m \le e \le M .$$

Finally, there is a sign $+$ or $-$ for each nonzero floating-point number.

Let $F = F(\beta, s, m, M)$ be the floating-point number system. Each nonzero $x \in F$ has the structure

$$x = \pm \, . \, d_1 d_2 \ldots d_s \cdot \beta^e ,$$

where the integers $d_1, \ldots, d_s$ have the bounds

$$1 \le d_1 \le \beta - 1 ,$$

$$0 \le d_i \le \beta - 1 \qquad (i=2,\ldots,s) ,$$

$$m \le e \le M .$$

Finally, the number 0 belongs to F, and is represented by

$$+ .00 \ldots 0 \cdot \beta^m .$$

Actual computer number systems may differ in detail from the ideal one discussed here, but the differences are only of secondary relevance for the fundamental problems of round off.

Typical floating-point systems in use correspond to the following values of the parameters:

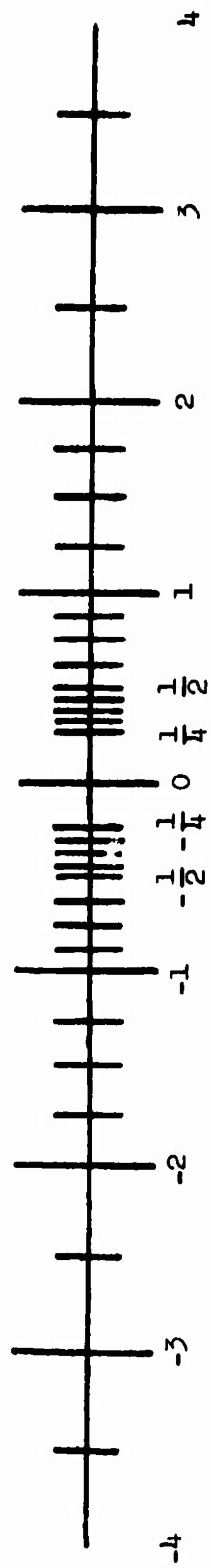
$\beta = 2$,	$s = 48$	$m = -1023$,	$M = 1024$	(Control Data 6600)
$\beta = 2$,	$s = 27$,	$m = -128$,	$M = 127$	(IBM 7090)
$\beta = 10$,	$s = 8$,	$m = -50$,	$M = 49$	(IBM 650)
$\beta = 8$,	$s = 13$,	$m = -51$,	$M = 77$	(Burroughs 5500)
$\beta = 16$,	$s = 6$,	$m = -64$,	$M = 63$	(IBM System/360)
$\beta = 16$,	$s = 14$,	$m = -64$,	$M = 63$	(IBM System/360)

Any one computer may be able to store numbers in more than one system. For example, the IBM System/360 uses the last two base-16 floating-point systems for scientific work, and also a certain base-10 system for accounting purposes.

F is not a continuum, nor even an infinite set. It has exactly $2(\beta-1)\beta^{s-1}(M - m + 1)+1$ numbers in it. These are not equally spaced throughout their range, but only between successive powers of β and their negatives. The accompanying figure, taken from [3], shows the 33-point set F for the small illustrative system $\beta = 2$, $s = 3$, $m = -1$, $M = 2$.

Because F is a finite set, there is no possibility of representing the continuum of real numbers in any detail. Indeed, real numbers in absolute value larger than the maximum member of F cannot be said to be represented at all. And, for many purposes, the same is true of real numbers smaller in magnitude than the smallest positive number in F . Moreover, each number in F has to represent a whole interval of real numbers. If x and y are two real numbers in the range of F , they will usually be represented by the same number in F whenever $|x-y|/|x| \leq \frac{1}{2} \beta^{-s}$; it is not important to be more precise here.

As a model of the real number system R , the set F has the arithmetic operations defined on it, as carried out by the digital computer. Suppose x and y are floating-point numbers. Then the true sum $x + y$ will frequently not be in F . (For example, take the 33-point system illustrated above, let $x = 5/4$ and $y = 3/8$.) Thus


-4
-3
-2
-1
-1/2
-1/4
0
1/4
1/2
1
2
3
4

the operation of addition, for example, must itself be simulated on the computer by an approximation called floating-point addition whose result will be denoted by $fl(x + y)$. Ideally, $fl(x + y)$ should be that member of F which is closest to the true $x + y$ (and either one, in case of a tie). In most computers this ideal is almost, but not quite, achieved. Thus in our toy 33-point set F we would expect that $fl(5/4 + 3/8)$ would be either 3/2 or 7/4 . The difference between $fl(x + y)$ and $x + y$ is called the rounding error in addition.

The reason that $5/4 + 3/8$ is not in the 33-point set F is related to the spacing of the members of F . On the other hand, a sum like $7/2 + 7/2$ is not in F because 7 is larger than the largest member of F . The attempt to form such a sum on most machines will cause a so-called overflow signal, and often the computation will be curtly terminated, for it is considered impossible to provide a useful approximation to numbers beyond the range of F .

While quite a number of the sums $x + y$ (for x, y in F) are themselves in F , it is quite rare for the true product $x.y$ to belong to F , since it will always involve 2s or 2s-1 significant digits. Moreover, overflow is much more probable in a product. Finally, the phenomenon of underflow occurs in floating-point multiplication, when two nonzero numbers x, y have a nonzero product that is smaller in magnitude than the smallest nonzero number in F . (Underflow is also possible, though unusual, in addition.) Thus the simulated multiplication operation, $fl(x.y)$, involves rounding even more often than floating addition.

The operations of floating addition and multiplication are commutative, but not associative, and the distributive law fails for them also. Since these algebraic laws are fundamental to mathematical analysis, working with floating-point operations is very difficult for mathematicians. One of the greatest mathematicians of the century, John von Neumann, was able to carry out some large analyses with floating-point arithmetic (see [10]), but they were extremely ponderous. Even his genius failed to discover a method of avoiding nonassociative analysis. Such a new method, called inverse error analysis, owes its origins to Cornelius Lanczos and Wallace Givens, and has been heavily exploited by J. H.Wilkinson. A detailed study of inverse error analysis is part of the subject of numerical analysis. We will mention it again in Section 5.

4. Two examples of round-off problems

One of the commonest functions of analysis is the exponential function e^x. Since it is so much used, it is essential to be able to have the value of e^x readily available in a computer program, for any floating-point number x. There is nowhere near enough storage to file a table of all values of e^x, so one must instead have an algorithm for recomputing e^x whenever it is needed. (By an algorithm we mean a process that is completely defined and guaranteed to terminate by delivering the desired output.) There are, in fact, a great many different methods such an algorithm could use, and most scientific computing systems have one programmed into it. But let us assume such an algorithm did not exist on your computer, and ask how you would program it. This is a realistic model of the situation for a more obscure transcendental function of analysis.

Recall that, for any real (or even complex) value of x, e^x can be represented as the sum of the universally convergent infinite series

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots .$$

Since you learned mathematics because it is useful, you will surely expect to use the series to compute e^x. Suppose that your floating-point number system F is characterized by $\beta = 10$ and $s = 5$. Let us use the series for $x = -5.5$, as proposed by Stegun and Abramowitz [13]. Here are the numbers we get:

$$
\begin{array}{rr}
e^{-5.5} = & 1.0000 \\
 & -5.5000 \\
 & +15.125 \\
 & -27.730 \\
 & +38.129 \\
 & -41.942 \\
 & +38.446 \\
 & -30.208 \\
 & +20.768 \\
 & -12.692 \\
 & +6.9803 \\
 & -3.4902 \\
 & +1.5997 \\
 & \vdots \\
\hline
 & +0.0026363
\end{array}
$$

The sum is terminated when the addition of further terms stops changing it, and this turns out to be after 25 terms. Is this a satisfactory algorithm? It may seem so, but in fact $e^{-5.5} = 0.00408677$, so that the above series gets an answer correct to only about 36 percent! This is awful.

What is wrong? Observe that there has been a lot of cancellation in forming the sum of this alternating series. Indeed, the four leading (i.e., most significant) digits of the eight terms that exceed 10 in modulus have all been lost. Professor D. H. Lehmer calls this phenomenon catastrophic cancellation, and it is fairly common in badly conceived computations. However, as Professor William Kahan has observed, this great cancellation is not the cause of the error in the answer -- it merely reveals the error. The error had already been made in that the terms like 38.129, being limited to 5 decimal digits, can have only one digit that contributes to the precision of the final answer. It would be necessary for the term $(-5.5)^4/4!$ to be carried to 8 decimals (i.e., 10 leading digits) for it to include all 6 leading digits of the answer. Moreover, an eleventh leading digit would be needed to make it likely that the sixth significant digit would be correct in the sum. The same is true of all terms over 10 in magnitude.

While it is usually possible to carry extra digits in a computation, it is always costly in time and space. For this particular problem there is a much better cure, namely, compute the sum for $x = 5.5$, and then take the reciprocal of the answer:

$$
\begin{aligned}
e^{-5.5} &= 1/e^{5.5} \\
&= 1/(1 + 5.5 + 15.125 + \dots) \\
&\doteq 0.0040865 \text{ , with our 5-decimal arithmetic.}
\end{aligned}
$$

(The symbol '$\doteq$' means 'equals approximately'.) With this computation, the error is reduced to 0.007 percent.

Note how much worse the problem would be if we wanted to compute e^x for $x = -100$.

Actual computer algorithms for calculating e^x often use a rational function of x , for x on a fairly short interval like $1 \leq x < e$. If x is outside this interval, say

$$e^a \leq x < e^{a+1} ,$$

then well known properties of the exponential function are used to obtain the answer from the rational approximation to e^y , where $y = x/e^a$. The creation of such algorithms for special functions is a branch of numerical analysis in which the general mathematician can hardly be an expert. On the other hand, it is part of the author's contention that mathematics books ought to mention the fact that a Taylor's series is often a very poor way to compute a function.

I will briefly state a second example. Recall from the calculus that

$$\text{(1)} \qquad \int_a^b \frac{dx}{x^p} = \left. \frac{x^{1-p}}{1-p} \right]_a^b = \frac{1}{1-p} (b^{1-p} - a^{1-p}) \qquad (p \neq 1) .$$

Now using a floating-point system with $\beta = 10$ and $s = 6$, let us evaluate the above formula for $a = 1$, $b = 2$, and $p = 1.0001$. We have

$$\text{(2)} \qquad I = \int_1^2 \frac{dx}{x^{1.0001}} = \frac{1 - 2^{-.0001}}{0.0001} .$$

If we use 6-place logarithms to evaluate $2^{-.0001}$, we have

$$\log_{10} 2 \doteq 0.301030 ,$$

$$\log_{10} 2^{-.0001} \doteq -0.0000301030 = -1 + 0.999969$$

whence, using our logarithm table again,

$$2^{-.0001} \doteq 0.99993 .$$

Thus, from (2), we get $I \doteq 0.7$, an answer correct to only one digit.

The precise meaning of the restriction to $\beta = 10$, $s = 6$ is not so clear in the evaluation of $2^{-.0001}$ as it would have been in the previous example. However, the example does illustrate the fact that formula (1), which is precisely meaningful for real numbers as long as $p \neq 1$, is difficult to use with finite-precision arithmetic for p close to 1 . Thus practical computation cannot admit the precise distinction between equality and inequality basic to pure mathematics. There are degrees of uncertainty caused by approximate equality.

5. Solving quadratic equations

The two examples of Section 4 were taken from the calculus. But we don't have to learn college mathematics to find algorithms. In ninth grade there is a famous algorithm for solving a quadratic equation, implicit in the following mathematical theorem:

Theorem. If a, b, c are real and $a \neq 0$, then the equation $ax^2 + bx + c = 0$ is satisfied by exactly two values of x, namely

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \tag{3}$$

and

$$x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a} \tag{4}$$.

Let us see how these formulas work when used in a straightforward manner to induce an algorithm for computing x_1 and x_2. This time we shall use a floating-point system with $\beta = 10$, $s = 8$, $m = -50$, $M = 50$; this has more precision than many widely used computing systems.

Case 1: $a = 1$, $b = -10^{-5}$, $c = 1$.

The true roots of the corresponding quadratic equation, correctly rounded to 11 significant decimals, are:

$x_1 = 99999.999990$ (true)

$x_2 = 0.000010000000001$ (true) .

If we use the expressions of the theorem, we compute

$x_1 = 100000.00$ (very good)

$x_2 = 0$ (100 percent wrong) .

(The reader is advised to be sure he sees how x_2 becomes 0 in this floating-point computation.)

Once again, in computing x_2 we have been a victim of catastrophic cancellation, which, as before, merely reveals the error we made in having chosen this way of computing x_2 . There are various alternate ways of computing the roots of a quadratic equation that do not force such cancellation. One of them follows from the easily proved formulas, true if $bc \neq 0$:

$$(5) \qquad x_1 = \frac{2c}{-b - \sqrt{b^2 - 4ac}} ,$$

$$(6) \qquad x_2 = \frac{2c}{-b + \sqrt{b^2 - 4ac}} .$$

Now, if $b < 0$, there is cancellation in (4) and (5) but not in (3) and (6). And, if $b > 0$, there is cancellation in (3) and (6), but not in (4) and (5). Special attention must be paid to cases where b or c is 0 .

At this point I would like to propose the following criterion of performance of a computer algorithm for solving a quadratic equation. This is stated rather loosely here, but a careful statement will be found in [2].

We define a complex number z to be <u>well within the range of F</u> if either $z = 0$ or

$$\beta^{m+2} \leq \mathrm{Re}(z) \leq \beta^{M-2} \quad \text{and} \quad \beta^{m+2} \leq \mathrm{Im}(z) \leq \beta^{M-2} .$$

This means that the real and imaginary parts of z are safely within the magnitudes of numbers that can be closely approximated by a member of F . The arbitrary factor β^2 is included to yield a certain margin of safety.

Suppose a , b , c are all numbers in F that are well within the range of F . Then they must be acceptable as input data to the quadratic equation algorithm. If $a = b = c = 0$, the algorithm should terminate with a message signifying that all complex numbers satisfy the equation $ax^2 + bx + c = 0$. If $a = b = 0$ and $c \neq 0$, then the algorithm should terminate with an error message that no complex number satisfies the equation.

Otherwise, let z_1 and z_2 be the exact roots of the equation, so numbered that $|z_1| \leq |z_2|$. (If $a = 0$, set $z_2 = \infty$.) Whenever z_1 is well within the range of F, the algorithm should determine an approximation that is close to z_1, in the sense of differing by not more than, say, $\beta + 1$ units in the least significant digit of the root. The same should be done for z_2.

If either or both of the roots z_i are not well within the range of F, then an appropriate message should be given, and the root (if any) that is well within the range of F should be determined to within a close approximation.

That concludes the loose specification of the desired performance of a quadratic equation solving algorithm. Let us return to a consideration of some typical equations, to see how the quadratic formulas work with them.

<u>Case 2</u>: $a = 6$, $b = 5$, $c = -4$.

There is no difficulty in computing $x_1 = 0.50000000$ and $x_2 = -1.3333333$, or nearly these values, by whatever formula is used.

<u>Case 3</u>: $a = 6 \times 10^{30}$, $b = 5 \times 10^{30}$, $c = -4 \times 10^{30}$.

Since the coefficients in Case 3 are those of Case 2, all multiplied by 10^{30}, the roots are unchanged. However, application of any of the formulas (3)-(6) causes overflow to occur very soon, since $b^2 > 10^{50}$, out of the range of F. Probably this uniform large size of $|a|$, $|b|$, $|c|$ could be detected before entering the algorithm, and all three numbers could be divided through by some scale factor like 10^{30} to reduce the problem to Case 2.

Case 4: $a = 10^{-30}$, $b = -10^{30}$, $c = 10^{30}$.

Here z_1 is near 1 , while z_2 is near 10^{60} . Thus our algorithm must determine z_1 very closely, even though z_2 is out of the range of F . Obviously any attempt to bring the coefficients to approximate equality in magnitude by simply dividing them all by the same number is doomed to failure, and might itself cause an overflow or underflow. This equation is, in fact, a severe test for a quadratic equation solver and even for the computing system in which the solver is run.

The reader may think that a quadratic equation with one root out of the range of F and one root within the range of F is a contrived example of no practical use. If so, he is mistaken. In many iteration algorithms which solve a quadratic equation as a subroutine, the quadratics do have a singular behavior in which $a \to 0$ as convergence occurs. One such example is Muller's method [9] for finding zeros of general smooth functions of z .

Case 5: $a = 1.0000000$, $b = -4.0000000$, $c = 3.9999999$.

Here the two roots are $z_1 = 1.999683772$, $z_2 = 2.000316228$. But applying the quadratic formulas (3), (4) gives

$$z_1 = z_2 = 2.0000000 \quad ,$$

with only the first four digits correct. These roots fail badly to meet my criteria, but the difficulty here is different from that in the other examples. The equation corresponding to Case 5 is the first of our equations in which a small relative change in a coefficient a , b , c induces a much larger relative change in the roots z_1 , z_2 . This is a form of instability in the equation itself, and not in the method of solving it. To see how unstable the problem is, the reader should show that the computed roots 2.0000000 are the exact roots of the equation

$$0.999999992x^2 - 3.999999968x + 3.999999968 = 0 \quad ,$$

in which the three coefficients differ, respectively, from the true a , b , c of Case 5 by less than one unit in the last significant digit. In this sense one can say that 2 , 2 are pretty good roots for Case 5.

This last way of looking at rounding errors is called the <u>inverse error approach</u> and has been much exploited by J. H. Wilkinson. In general, it is characterized by asking how little change in the data of a problem would be necessary to cause the computed answers to be the exact solution of the changed problem. The more classical way of looking at round off, the <u>direct error approach</u>, simply asks how wrong the answers are as solutions of the problem with its given data. While both methods are useful, the important feature of inverse error analysis is that in many large matrix or polynomial problems, it can permit us easily to continue to use associative operations, and this is often very difficult with direct error analysis.

Despite the elementary character of the quadratic equation, it is probably still safe to say that not more than five computer algorithms exist anywhere that meet the author's criteria for such an algorithm. Creating such an algorithm is not a very deep problem, but it does require attention to the goal and to the details of attaining the goal. It illustrates the sort of place that an undergraduate mathematics or computer science major can make a substantial contribution to computer libraries.

I wish to acknowledge that the present section owes a great deal to lectures by Professor William Kahan of the University of California, Berkeley, given at Stanford in the Spring of 1966.

6. Solving linear systems of equations

As the high school student moves from ninth grade on to tenth or eleventh, he will encounter the solution of systems of linear algebraic equations by Gauss' method of eliminating unknowns. With a little systematization, it becomes another algorithm for general use. I would like to examine it in the simple case of two equations in two unknowns, carried out on a computer with $\beta = 10$, $s = 3$.

Let the equation system be one treated by Forsythe and Moler [3]:

$$(7) \quad \begin{cases} 0.000100x + 1.00y = 1.00 \\ 1.00x \quad + 1.00y = 2.00 \end{cases} .$$

The true solution, rounded correctly to the number of decimals shown, is

$$x = 1.00010 \quad , \quad y = 0.99990 \qquad \text{(truly rounded)}.$$

The Gauss elimination algorithm uses the first equation (if possible) to eliminate the first variable, x , from the second equation. This is done by multiplying the first equation by 10000 , and subtracting it from the second equation. When we work to three significant digits, the resulting system takes the form

$$\begin{cases} 0.000100x + \quad 1.00y = 1.00 & \text{(the old first equation)} \\ \qquad\qquad - 10000 \quad y = -10000 & . \end{cases}$$

For just two equations, this completes the elimination of unknowns. Now commences the back solution. One solves the new second equation for y , finding that $y = 1.00$. This value is substituted into the first equation, which is then solved for x . One then finds $x = 0.00$. In summary, we have found

$$\begin{cases} y = 1.00 \\ x = 0.00 \end{cases} .$$

PRECEDING PAGE BLANK

Of course, this is awful! What went wrong? There was certainly no long accumulation of round-off errors, such as might be feared in a large problem. Nor was the original problem unstable of itself, as it would be if the lines represented by the two equations (7) were nearly parallel.

There is one case in which it is impossible to eliminate x from the second equation -- when the coefficient of x in the first equation is exactly 0. Were such an exact 0 to occur, the algorithm is preceded by interchanging the equations. Now, once again, if an exact zero makes a mathematical algorithm impossible, we should expect that a near zero will give a floating-point algorithm some kind of difficulty. That is a sort of philosophical principle behind what went wrong. And, in fact, the division by the nearly zero number 0.0001 introduced some numbers (10000) that simply swamped the much smaller, but essential data of the second equation. That is what went wrong.

How could this be avoided? The answer is simple, in this case. If it is essential to interchange equations when a divisor is actually zero, one may suspect that it would be important, or at least safer, to interchange them when the coefficient of x in the first equation is much smaller in magnitude than the coefficient of x in the second equation. A careful round-off analysis given by J. H. Wilkinson [14] proves this to be the case, and good linear equation solvers will make the interchange whenever necessary to insure that the largest coefficient of x (in magnitude) is used as the divisor. Thus the elimination yields the system

$$\begin{cases} 1.00x + 1.00y = 2.00 \\ \qquad\quad\; 1.00y = 1.00 \end{cases} .$$

After the back solution we find

$$\begin{cases} y = 1.00 \\ x = 1.00 \end{cases} ,$$

a very fine result.

This algorithm, with its interchanges, can be extended to n equations in n unknowns, and is a basic algorithm found in all good computing centers.

The following example shows that there remains a bit more to the construction of a good linear equation solver. Consider the system

$$(8) \qquad \begin{cases} 10.0\,x + 100000\,y = 100000 \\ 1.00x + 1.00y = 2.00 \quad . \end{cases}$$

If we follow the above elimination procedure, we see that interchanging the equations is not called for, since $10.0 > 1.00$. Thus one multiplies the first equation by 0.100 and subtracts it from the second. One finds afterwards, still working with $\beta = 10$, $s = 3$, that

$$\begin{cases} 10.0x + 100000y = 100000 \\ \quad - 10000y = -10000 \quad . \end{cases}$$

Back solving, one finds

$$\begin{cases} y = 1.00 \\ x = 0.00 \quad ! \end{cases}$$

This is just as bad as before, for system (8) has the same solution as (7). Indeed, system (8) is easily seen to be identical with (7), except that the first equation has been multiplied through by 100000 .

So, the advice to divide by the largest element in the column of coefficients of x is not satisfactory for an arbitrary system of equations. What seems to be wrong with the system (8) is that the first equation has coefficients that are too large for the problem. Before entering the Gaussian elimination algorithm with interchanges, it is necessary to scale the equations so that the coefficients are roughly of the same size in all equations. This concept of scaling is not completely understood as yet, although in most practical problems we are able to do it well enough.

If you were faced with having to solve a nonsingular system of linear algebraic equations of order 26, for example, you might wonder how to proceed. Some mathematics books express the solution by Cramer's rule, in which each of the 26 components is the quotient of a different numerator determinant by a common denominator determinant. If you looked elsewhere, you might find that a determinant of order 26 is the sum of 26! terms, each of which is the product of 26 factors. If you decide to proceed in this manner, you are going to have to perform about $25 \times 26!$ multiplications, not to mention a similar number of additions. On a fast contemporary machine, because of the time required to do preparatory computations, you would hardly perform more than 100,000 multiplications per second. And so the multiplications alone would require about 10^{17} years, if all went well. The round-off error would usually be astronomical.

In fact, the solution can be found otherwise in about $(1/3) \times 26^3 \doteq 5859$ multiplications and a like number of additions, and should be entirely finished in under half a second, with very little round-off error. So it can pay to know how to solve a problem.

I wish to leave you with the feeling that there is more to solving linear equations than you may have thought.

7. When do we have a good solution?

Another example of a linear algebraic system has been furnished by Moler [8]:

$$
(9) \qquad \begin{cases} 0.780x + 0.563y - 0.217 = 0 \\ 0.913x + 0.659y - 0.254 = 0 \ . \end{cases}
$$

Some one proposes two different solutions to (9), namely

$$(x_1, y_1) = (0.999, -1.001)$$

and

$$(x_2, y_2) = (0.341, -0.087) \ .$$

Which one is better? The usual check would be to substitute them both into (9). We obtain

$$
\begin{cases} 0.780x_1 + 0.563y_1 - 0.217 = -0.001243 \\ 0.913x_1 + 0.659y_1 - 0.254 = -0.001572 \end{cases}
$$

and

$$
\begin{cases} 0.780x_2 + 0.563y_2 - 0.217 = -0.000001 \\ 0.913x_2 + 0.659y_2 - 0.254 = 0 \end{cases} \ .
$$

It seems clear that (x_2, y_2) is a better solution than (x_1, y_1), since it makes the residuals far smaller.

However, in fact the true solution is $(1, -1)$, as the reader can verify easily. Hence (x_1, y_1) is far closer to the true solution than (x_2, y_2) !

A persistent person may ask again: which solution is really better? Clearly the answer must depend on one's criterion of goodness: a small residual, closeness to the true solution, or perhaps something else. Surely one will want different criteria for different problems. The pitfall to be avoided here is the belief that all such criteria are necessarily satisfied, if one of them is.

8. Sensitivity of certain problems

We now show that certain computational problems are surprisingly sensitive to the data. This aspect of numerical analysis is independent of the floating-point number system.

We first consider the zeros of polynomials in their dependence on the coefficients. In Case 5 of Section 4 above, we noted that, while the polynomial $x^2 - 4x + 4$ has the double zero $2, 2$, the rounded roots of the polynomial equation

$$x^2 - 4x + 3.9999999 = 0 \tag{10}$$

are 1.999683772 and 2.000316228. Thus the change of just one coefficient from 4 to 3.9999999 causes both roots to move a distance of $.000316228$. The displacement in the root is 3162 times as great as the displacement in the coefficient.

The instability just described is a common one, and results from the fact that the square root of ϵ is far larger than ϵ. For the roots of (10) are the roots of

$$(x - 2)^2 = \epsilon \quad , \quad \epsilon = .0000001 \quad ,$$

and these are clearly $2 \pm \sqrt{\epsilon}$. For equations of higher degree, a still more starting instability would have been possible.

However, it is not only for polynomials with nearly multiple zeros that instability can be observed. The following example is due to Wilkinson [14]. Let

$$\begin{aligned} p(x) &= (x + 1)(x + 2)\dots(x + 19)(x + 20) \\ &= x^{20} + 210x^{19} + \dots \end{aligned} \quad .$$

The zeros of $p(x)$ are $-1, -2, \dots, -19, -20$, and are well separated. This example evolved at a place where the floating-point number system had $\beta = 2$, $s = 30$. To enter a typical coefficient into the computer, it is necessary to round it to 30 significant base-2 digits. Suppose that a change in the 30-th most significant base-2 digit is made in only one of the twenty coefficients. In fact, suppose that the coefficient of x^{19} is changed from 210 to $210 + 2^{-23}$. How much effect does this small change have on the zeros of the polynomial?

To answer this, Wilkinson carefully computed $(\beta = 2 , s = 90)$ the roots of the equation $p(x) + 2^{-23}x^{19} = 0$. These are now listed, correctly rounded to the number of digits shown.

1.00000 0000	10.09526 6145 $\pm$ 0.64350 0904i
2.00000 0000	11.79363 3881 $\pm$ 1.65232 9728i
3.00000 0000	13.99235 8137 $\pm$ 2.51883 0070i
4.00000 0000	16.73073 7466 $\pm$ 2.81262 4894i
4.99999 9928	19.50243 9400 $\pm$ 1.94033 0347i
6.00000 6944	
6.99969 7234	
8.00726 7603	
8.91725 0249	
20.84690 8101	

Note that the small change in the coefficient 210 has caused ten of the zeros to become complex, and that two have moved more than 2.81 units off the real axis! Of course, to enter $p(x)$ completely into the computer would require many more roundings, and actually computing the zeros could not fail to cause still more errors. The above table of zeros was produced by a very accurate computation, and does not suffer appreciably from round-off errors. The reason these zeros moved so far is not a round-off problem -- it is a matter of sensitivity. Clearly zeros of polynomials of degree 20 with well separated zeros can be much more sensitive to changes in the coefficients than you might have thought.

To motivate a second example, let me quote a standard theorem of algebra: In the ring of square matrices of fixed order n , if $AX = I$, where I is the identity matrix of order n , then $XA = I$.

It follows from this theorem and continuity considerations that, if A is a fixed matrix and X a variable one, and if $AX - I \to \Theta$, the zero matrix, then also $XA - I \to \Theta$. Hence, if $AX - I$ is small in some sense, then $XA - I$ is also small. However, as with polynomials, one's intuition may not be very good at guessing how small these smallnesses are. Here is an example: Fix

$$A = \begin{bmatrix} 9999 & 9998 \\ 10000 & 9999 \end{bmatrix} .$$

Let

$$X = \begin{bmatrix} 9999.9999 & -9997.0001 \\ -10001 & 9998 \end{bmatrix} .$$

Then a computation without round-off shows that

$$AX - I = \begin{bmatrix} .001 & .0001 \\ 0 & 0 \end{bmatrix} .$$

From the last equality the reader may conclude that X is close, though not equal, to the unique inverse A^{-1}. However, another calculation without round off shows that

$$XA - I = \begin{bmatrix} 19997.0001 & 19995.0003 \\ -19999 & -19995 \end{bmatrix} .$$

Thus the quantities $AX - I$ and $XA - I$, which must vanish together, can be of enormously differing magnitudes in a sensitive situation, even for matrices of order 2.

The true inverse matrix is given by

$$A^{-1} = \begin{bmatrix} 9999 & -9998 \\ -100000 & 9999 \end{bmatrix} ,$$

and this is hardly close to X.

9. A least-squares problem of Hilbert

The following least-squares problem was discussed by the great mathematician David Hilbert [6], and leads to some interesting matrices. Fix $n \geq 1$. Let $f(t)$ be given and continuous for $0 \leq t \leq 1$. We wish to approximate $f(t)$ as well as we can by a polynomial $x_0 + x_1 t + x_2 t^2 + \ldots + x_{n-1} t^{n-1}$ of degree $n - 1$. To be more precise, we wish to determine $x_0, x_1, \ldots, x_{n-1}$ so that

$$\Phi(x) = \int_0^1 [f(t) - x_0 - x_1 t - \ldots - x_{n-1} t^{n-1}]^2 \, dt$$

is as small as possible. It is not difficult to show that the minimizing vector of coefficients x exists, is unique, and can be determined by solving the system of n simultaneous equations

$$(11) \qquad \frac{\partial \Phi}{\partial x_i} = 0 \qquad (i = 0, 1, \ldots, n-1) \; .$$

If you carry out the algebra, you find that (11) is equivalent to the system of n linear algebraic equations

$$(12) \qquad Ax = b \; ,$$

where

$$(13) \qquad a_{i,j} = \int_0^1 t^{i-1} t^{j-1} dt = \frac{1}{i+j-1} \qquad (i, j = 1, 2, \ldots, n)$$

and

$$(14) \qquad b_i = \int_0^1 t^{i-1} f(t) dt \qquad (i = 1, 2, \ldots, n) \; .$$

The matrix A of coefficients in (12) is now called the Hilbert matrix (of order n), and is denoted by H_n :

$$
H_n = \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \cdots & \frac{1}{n} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \cdots & \frac{1}{n+1} \\ \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & & \cdot \\ \frac{1}{n} & \frac{1}{n+1} & \frac{1}{n+2} & \cdots & \frac{1}{2n-1} \end{bmatrix} .
$$

The equations (12) with matrix $A = H_n$ are called the <u>normal equations</u> for the problem. It appears that all one has to do is to find and use a quadrature rule for approximating the b_i in (14), and then solve the system (12). This is certainly the standard advice in books on practical statistics.

However, what is observed is that for n bigger than 8 or 9 (the threshold depends on the system used), linear equations solvers in ordinary floating-point precision will simply refuse to solve (12). Moreover, for problems that can be solved (say $n = 6$), there are enormous differences in the solution vectors x for apparently identical problems on slightly different machines. Why all this trouble?

Let me try to explain the sensitivity of the problem first. Let $T_n = H_n^{-1}$. Then it can be proved that

$$
T_6 = \begin{bmatrix} 36 & -630 & 3360 & -7560 & 7560 & -2772 \\ -630 & 14700 & -88200 & 211680 & -220500 & 83160 \\ 3360 & -88200 & 564480 & -1411200 & 1512000 & -582120 \\ -7560 & 211680 & -1411200 & 3628800 & -3969000 & 1552320 \\ 7560 & -220500 & 1512000 & -3969000 & 4410000 & -1746360 \\ -2772 & 83160 & -582120 & 1552320 & -1746360 & 698544 \end{bmatrix} .
$$

This means that a change of 10^{-6} in just the one element b_5 will produce changes in the solution vector x of

$$
(.00756, -.2205, 1.512, -3.969, 4.41, -1.74636)^T .
$$

Such changes are unavoidable in a system with $\beta = 10$ and $s = 7$. This means that some of the coefficients of the best fitting polynomial of degree 5 will have unavoidable uncertainties of the order of 4 units. This may give some explanation of the instability in the answers. More details are in Section 19 of [3].

Here are approximate values of t_n, the maximum elements in T_n, for $n \leq 10$:

n	t_n
2	1.20×10^1
3	1.92×10^2
4	6.48×10^3
5	1.79×10^5
6	4.41×10^6
7	1.33×10^8
8	4.25×10^9
9	1.22×10^{11}
10	3.48×10^{12}

It cannot be demonstrated here, but if $t_n \gg \beta^s$, you just cannot solve the system $H_n x = b$ with s-digit arithmetic to base β.

The conclusion of this example is that one should not follow a Statistics book blindly here. It is much better to arrange things so that matrices of Hilbert type do not arise, even approximately. And when they do, one must be sure to use enough precision so that $t_n \ll \beta^s$. There are other ways of attacking least-squares problems which are less sensitive to the data.

10. Instability in solving ordinary differential equations

The standard initial-value problem for a single ordinary differential equation $dy/dx = f(x, y)$ is to determine $y(x)$ as accurately as possible for $x > 0$, given $y(0)$. In one very common class of methods (the multistep methods) of solving this problem approximately, one picks a fixed interval $h > 0$, and determines y_n to approximate $y(nh)$ for $n = 1, 2, \ldots$. One highly recommended multistep method in desk-computing days was the Milne-Simpson method. Here one let $y_0 = y(0)$, the given initial value, and determined y_1 by some method not mentioned here. Let $y'_n = f(nh, y_n)$. The idea was to determine y_{n+1} from y_{n-1} and y_n $(n = 1, 2, \ldots)$ by the integral

$$y_{n+1} = y_{n-1} + \int_{(n-1)h}^{(n+1)h} f(x, y(x))dx \quad . \tag{15}$$

Since the integral in (15) can not usually be evaluated exactly, Milne's idea was to approximate it by Simpson's formula, and so let

$$y_{n+1} = y_{n-1} + \frac{h}{3}(y'_{n-1} + 4y'_n + y'_{n+1}) \quad . \tag{16}$$

At the time we seek to find y_{n+1} from (16) we know y_{n-1} and y_n, and hence y'_{n-1} and y'_n; but y'_{n+1} is not known. For general f, Milne [7] determined the solution of (16) by an iterative process that is irrelevant to the present discussion. Let us merely assume that y_{n+1} has been found so that (16) holds, where $y'_{n+1} = f((n+1)h, y_{n+1})$, and that this has been done for $n = 1, 2, \ldots$, as far as we wish to go. This method was highly recommended by Milne for solution of ordinary differential equations at a desk calculator, and it seemed to work very well indeed. Most problems were probably solved within 30 steps or less.

As soon as automatic digital computers arrived on the scene, users of the Milne-Simpson method started to find extraordinary behavior in certain problems. To illustrate what happened, let us take the very simple test problem

$$dy/dx = f(x, y) = -y \quad , \quad \text{with} \quad y(0) = 1 \quad .$$

The true solution, of course, is $y = e^{-x}$.

Take $h = 0.1$, and carry out the Milne-Simpson process with $y_0 = 1$ and $y_1 = 0.90483742$, an 8-decimal correctly rounded value of $e^{-0.1}$. Thi is not something you can do in your head, and so I will give you the results, as computed on a system with $\beta = 10$, $s = 8$.

x	$y_{computed}$	e^{-x}
.2	.81873069	.81873075
.3	.74081817	.74081822
⋮	⋮	⋮
8.0	.00033519912	.00033546263
8.1	.00030380960	.00030353914
⋮	⋮	⋮
13.2	.0000036689301	.0000018506012
13.3	.0000032084360	.0000016744932
13.4	-.00000070769248	.0000015151441
⋮	⋮	⋮

We see that by $x = 8.0$ a noticeable oscillation has set in, whereby successive values of y_n alternate in being too low and too high. By $x = 13.4$ this oscillation has grown so violent that it has (for the first time) actually thrown the sign of y_n negative, which is unforgiveable in anything simulating a real exponential function!

The Milne-Simpson method is very accurate, in that the Simpson formula is an accurate approximation to the above integral. What can be the matter?

Since $f(x,y) = -y$, we can explicitly write down the formula (16) in the form

$$y_{n+1} = y_{n-1} - \frac{h}{3}(y_{n-1} + 4y_n + y_{n+1}) \quad .$$

Thus the computed $\{y_i\}$ satisfy the 3-term recurrence relation

$$(1 + \frac{h}{3})y_{n+1} + \frac{4h}{3} y_n - (1 - \frac{h}{3})y_{n-1} = 0 . \tag{17}$$

We know that the general solution of (17) takes the form

$$y_n = A_1\lambda_1^n + A_2\lambda_2^n , \tag{18}$$

where λ_1, λ_2 are the roots of

$$(1 + \frac{h}{3})\lambda^2 + \frac{4h}{3} \lambda - (1 - \frac{h}{3}) = 0 . \tag{19}$$

Some algebra and elementary analysis show that

$$\lambda_1 = 1 - h + 0(h^2) \quad , \quad \text{as} \quad h \to 0 ,$$

$$\lambda_2 = -(1 + \frac{h}{3}) + 0(h^2) \quad , \quad \text{as} \quad h \to 0 .$$

Putting the values of λ_1, λ_2 into (18), and using the relation $nh = x$, we find that, for small h,

$$\begin{aligned} y_n &\doteq A_1(1 - h)^n + (-1)^n A_2(1 + \frac{h}{3})^n \\ &= A_1(1 - h)^{\frac{1}{h}x} + (-1)^n A_2(1 + \frac{h}{3})^{\frac{3}{h} \cdot \frac{x}{3}} \\ &\doteq A_1e^{-x} + (-1)^n A_2e^{x/3} . \end{aligned}$$

The first term is the desired solution, and the second is an unwelcome extra solution of the difference equation (17) of the Milne-Simpson method. Now the initial conditions might have been chosen exactly so that $A_1 = 1$ and $A_2 = 0$. (They were roughly of this nature.) Had they been so chosen, and if the solution could have proceeded without round-off error, the unwanted term in A_2 would never have appeared. But, in fact, a small amount of this solution was admitted by the initial condition, and some more of it crept in as the result of round-off. Then, after enough steps, the size of $e^{x/3}$ caused the unwanted term to dominate the solution, with its oscillating sign.

This disaster never occurred in desk computation, so far as we know, because at a desk one just doesn't carry out enough steps. However, Professor Milne tells me that he did occasionally observe harmless oscillations in the low-order digits.

The moral of this example is that not only are math books not enough, but even old numerical analysis books are not enough to keep you out of some pitfalls!

11. Instability in solving a partial differential equation

The following is a simple problem for the heat equation. Suppose a homogeneous insulated rod of length 1 is kept at temperature 0 at one end, and at temperature 1 at the other end. If the entire rod is initially at temperature 0 , how does it warm up?

Let $u = u(x, t)$ denote the temperature at time t at that part of the rod that is x units from the cold end. Then, if the units were chosen to make the conductivity 1 , the temperature u satisfies the differential equation

$$\frac{\partial^2 u}{\partial x^2} = \frac{\partial u}{\partial t} \qquad (0 < x < 1 \; ; \; t > 0) \; , \tag{20}$$

with end and initial conditions

$$\left\{ \begin{array}{ll} u(0, t) = 0 & (t > 0) \; , \\ u(1, t) = 1 & (t > 0) \; , \\ u(x, 0) = 0 & (0 < x < 1) \; . \end{array} \right. \tag{21}$$

This problem can perhaps best be solved by separation of variables and trigonometric series. But let us apply the method of finite differences, which might in any case be needed for a more difficult problem. To do this, we divide the length of the rod into equal intervals, each of length h . And we divide the time interval $[0, \infty)$ into equal intervals of length k . Instead of trying to determine $u(x, t)$ for all x and t , we will limit ourselves to computing $u(x, t)$ on the discrete net of points of type (mh, nk) , for integers m, n . The heat equation (20) can then be simulated by a number of finite-difference equations, of which we pick one:

$$\frac{u(x-h, t) - 2u(x, t) + u(x+h, t)}{h^2} = \frac{u(x, t+k) - u(x, t)}{k} \; . \tag{22}$$

Equation (22) can be used to determine $u(x, t)$ for all net points in the infinite strip of the problem, as follows: Solve (22) for

$u(x, t+k)$ in terms of $u(x-h, t)$, $u(x, t)$, $u(x+h, t)$. Thus compute $u(x, k)$ for $x = h, 2h, \ldots, (n-1)h$ in terms of the given initial conditions on the line $t = 0$. The given end conditions give $u(0, k)$ and $u(1, k)$. With this set of values of u at all points of the net with $t = k$, we can continue and compute all values on the net for $t = 2k$. Etc. The computation is very attractive, because each new value of $u(x, t+k)$ is determined explicitly from (22) -- there is no need to solve a large number of simultaneous equations.

How does the solution behave? To try a case, we pick $h = 0.1$ and $k = 0.01$. Thus the rod is represented by 9 interior points and two endpoints, and we get a solution at time steps 0.01 apart. Just to show the behavior of the solution of (22), we give the value of the temperature $u(0.5, t)$ at the midpoint of the rod, computed with $\beta = 10$, $s = 8$, for selected times:

t	u(0.5, t) computed from k = 0.01
0	0
...	...
0.05	1
0.06	-4
0.07	16
...	...
0.15	132276
...	...
0.20	-28157050
...	...
0.99	$+1.0196022 \times 10^{44}$
1.00	$-2.9590007 \times 10^{44}$.

The values in the table are ridiculous, of course. It is a classical example of instability. Common sense and mathematics both tell us that the real temperature can never get outside the range $0 \leq u(x, t) \leq 1$. Our difference-equation problem is a disastrous model of the continuous problem, even though both difference expressions in (22) are reasonable models of the derivatives in (20).

This terrible pitfall has been known for at least 20 years, and yet new problem solvers keep on rediscovering it.

It is interesting to note that if one selects a time step only half as long, the computation proceeds very nicely. Here is the corresponding table of values of $u(0.5, t)$ for a computation ($\beta = 10$, $s = 8$) with $h = 0.1$, $k = 0.005$:

t	u(0.5, t) computed for k = 0.005
0	0
...	...
0.05	.10937500
0.06	.14599609
0.07	.17956543
...	...
0.15	.35637261
...	...
0.20	.41304382
...	...
1.00	.49997173

The values of the midpoint temperature are converging to 0.5, as they obviously should in the physical problem.

What is the reason for the great difference in behavior between $k = 0.005$ and $k = 0.01$? The matter can be analyzed in many ways, and here is one simple approach. Let $\lambda = k/h^2$. Then, from (22),

$$(23) \qquad u(x, t+k) = \lambda u(x-h, t) + (1-2\lambda)u(x, t) + \lambda u(x+h, t) \quad .$$

Hence, if $0 < \lambda \leq \frac{1}{2}$, the formula (23) represents $u(k, t+k)$ as a weighted average with non-negative weights of $u(x-h, t)$, $u(x, t)$, and $u(x+h, t)$. Hence $u(x, t+k)$ will always be between the maximum and minimum values of $u(x, t)$. But, if $\lambda > \frac{1}{2}$, the weights alternate in sign and thus permit a solution in which

$$|u(x, t+k)| = \lambda|u(x-h, t)| + (2\lambda-1)|u(x, t)| + \lambda|u(x+h, t)| .$$

Here the sum of the weights is $4\lambda - 1 > 1$. This permits an exponential growth of a solution with an alternating sign pattern.

Thus the condition $0 < \lambda = k/h^2 \leq \frac{1}{2}$ is essential to keep the solution bounded. A deeper discussion found, for example, in Forsythe and Wasow [4] proves that the solution of (22) converges to the solution of (20) uniformly for all (x, t) with $0 \leq x \leq 1$, $0 < t \leq T < \infty$, as $h \to 0$, $k \to 0$ in such a way that $k/h^2 \leq 1/2$.

The proof of convergence and an analysis of the stability of (22) can be carried out by means of Fourier analysis. The stability can be examined in more detail by studying the eigenvalues and eigenvectors of the linear transformation (23) that maps each line of solutions onto the next line.

Note that in our two tables we had $\lambda = 1$ and $\lambda = 1/2$, respectively.

12. Round-off errors in polynomial deflation

Our final example, due to Wilkinson [14], shows a more subtle effect of round-off error that arises in the course of finding polynomial zeros. The quartic polynomial

$$P_4(x) = x^4 - 6.7980x^3 + 2.9948x^2 - 0.043686x + 0.000089248$$

has zeros that, correctly rounded, are as follows:

$$0.0024532 \quad , \quad 0.012576 \quad , \quad 0.45732 \quad , \quad 6.32565 \quad .$$

I. Suppose first that we compute the zero 0.0024532 , and then deflate P_4 to a cubic by dividing $P_4(x)$ by $x - 0.0024532$, using $\beta = 10$, $s = 5$. If we do, the resulting cubic has zeros

$$0.012576 \quad , \quad 0.457315 \, , \quad 6.32561 \quad ,$$

so that the main error introduced by this deflation is a change of the largest zero by 4 units in its last place.

II. Suppose, on the other hand, that we first compute the zero 6.3256 , and then deflate P_4 to a cubic by dividing $P_4(x)$ by $x - 6.3256$, again using 5-place decimal arithmetic. If so, the resulting cubic has the zeros

$$0.0026261 \pm 0.064339\, i \ ,$$
$$0.467148 \quad .$$

We have perturbed two of the remaining zeros beyond recognition, and have changed the second significant digit of the third.

Thus it appears to matter a great deal which zero of P_4 we locate first. For the present case we can get a feeling for what is happening by examining the process of division of $P_4(x)$ by the linear factors. We use detached coefficients:

First, the division by $x - 0.0024532$:

$$
\begin{array}{lllll}
1 & -\ 6.7980 & +\ 2.9948 & -\ 0.043686 & +\ 0.000089248 \\
 & -\ 0.0024532 & +\ 0.166707206 & -\ 0.00730587492 & +\ 0.000089247416 \\
\hline
1 & -\ 6.7955 & +\ 2.9781 & -\ 0.036380 &
\end{array}
$$

Thus the cubic that results from the first deflation is $\tilde{P}_3(x) = x^3 - 6.7955x^2 + 2.9781x - 0.036380$. Moreover, a careful examination of the division shows that $\tilde{P}_3(x)$ is <u>exactly</u> (i.e., without round off) equal to the quotient of

$$\tilde{P}_4(x) = x^4 - 6.7979532x^3 + 2.9947707206x^2 - 0.04368587492x + 0.000089247416$$

by $x - 0.0024532$. Hence the zeros of $\tilde{P}_3$ are exactly the zeros of $\tilde{P}_4$ except for 0.0024532 . Note that all the coefficients of $\tilde{P}_4$ and P_4 are quite close, so it is reasonable to expect that the zeros of P_4 and $\tilde{P}_4$ should be close (as they are).

Now we show the deflation by $x - 6.3256$:

$$
\begin{array}{lllll}
1 & -\ 6.7980 & +\ 2.9948 & -\ 0.043686 & +\ 0.000089248 \\
 & -\ 6.3256 & +\ 2.98821344 & -\ 0.04174896 & +\ 0.0122526872 \\
\hline
1 & -\ 0.4724 & +\ 0.0066 & -\ 0.001397 &
\end{array}
$$

Thus the result of this deflation is a cubic $\hat{P}_3(x) = x^3 - 0.4724x^2 + 0.0066x - 0.001397$. Again, $\hat{P}_3(x)$ is exactly the quotient of

$$\hat{P}_4(x) = x^4 - 6.7980x^3 + 2.99481344x^2 - 0.04368596x + 0.0122526872$$

by $x - 6.3256$. Note that P_4 and $\hat{P}_4$ differ very much in their constant terms. Hence the product of the roots of $\hat{P}_4$ must be very different from that for P_4 . This is an explanation for the great shift of the zeros of $\hat{P}_3$.

Further analysis shows that the shift in zeros during this kind of deflation is generally small when deflation is made with zeros of small modulus, and is generally large when deflation is based on zeros of large modulus. Thus it is better to get zeros of small modulus first in using a polynomial solver with deflation in the above manner.

Of course, any zero of a deflated polynomial can be refined by use of the original polynomial, and that is normally done. But, zeros that change as much as those above are difficult to refine, since the refinement process may converge to the wrong zero.

13. Conclusions

Around ten years ago, when I last read a number of them, most mathematics books that dealt with numerical methods at all were from ten to fifty years out of date. In the past ten years, many excellent new methods have been devised for most of the elementary problems -- methods that are well adapted to automatic computers, and work well. Let me cite a few examples of important algorithms hardly known ten years ago:

1. For getting eigenvalues of stored square matrices, there is an excellent method that starts with the transformation of Householder (1958), and follows it with the QR-algorithm of Francis (1961-62) and Kublanovskaja (1961). It is the method of choice for most problems. For references, see Wilkinson [15].

2. For solving ordinary differential equations, special methods have been developed by Gear [5], Osborne [11], and others which can deal with so-called stiff equations. (Roughly speaking, a stiff equation is one whose solutions contain very rapidly decaying transients which contribute nothing to the long-term solution, but which interfere drastically with most numerical methods of solving the equation.)

3. For evaluating the definite integral of a smooth function of one real variable, the method of Romberg (see Vol. 2 of Ralston and Wilf [12]) has proved to be very useful.

4. For minimizing a smooth real-valued function of n real variables, a variant by Fletcher and Powell [1] of a method of Davidon is far superior to anything used in the 1950's. And there are still more recent methods.

Many other examples could be given. Indeed, the 1960's have proved almost explosive in the number of newly invented algorithms that have supplanted those known earlier. Of the methods known years ago for common numerical problems, only Gauss' systematic elimination method for solving linear algebraic equation systems with dense, stored matrices remains supreme today, and even it must be augmented with scaling and pivoting decisions, as we noted in Section 6 above. Newton's method for solving a nonlinear system of equations is still much used today, though it has strong competition from newer methods.

Because of my knowledge of mathematics texts ten years ago, and my knowledge of the explosive increase in numerical methods in the 1960's, I am confident that today's mathematics courses cannot be trusted to include important knowledge about computer methods. As we noted in Section 10 above, you can't trust early numerical analysis textbooks either.

On the other hand, there are experts in numerical analysis. They have societies in which methods are presented and discussed. The Society for Industrial and Applied Mathematics (SIAM) and the Special Interest Group on Numerical Mathematics (SIGNUM) of the Association for Computing Machinery (ACM) are the most active in this country. There are a number of journals with important information. For a start, you might consult the keyword-in-context index of Computing Reviews, the review journal published by ACM, as well as the algorithms in the Communications of ACM and in Numerische Mathematik. Modern monographs and textbooks in numerical analysis are slowly appearing, and the beginner might profitably consult Ralston and Wilf [12].

It might be noted as a digression that, just as mathematics departments mainly ignore modern numerical analysis, so also the newly created computer science departments often give the subject little attention, since they are so busy with a variety of important nonnumerical fields. Thus numerical analysts remain a small corps of specialists whose greatest appreciation probably comes from the users of mathematical programs.

Students of mathematics are well equipped to read about numerical methods. Why should they repeat the classical blunders of generations past? Why aren't they informed of the existence of good numerical methods, and roughly where to find them?

Remembering that most students take mathematics in order to apply it on computers, I ask why mathematics courses shouldn't reflect a true awareness of how computing is done? Why shouldn't students demand in their mathematics courses a greater awareness of the points of contact of pure mathematics and its practice on a computer?

Of course, a mathematics instructor can shrug his shoulders and say that actual computing problems don't interest him, and suggest that his students contact a numerical analyst sometime. If the instructor actually says this out loud, it at least has the virtue that the students may realize immediately that the mathematics is not applicable directly, instead of having to discover it for themselves. It still sounds irresponsible to me. After all, Society has been supporting mathematicians pretty well for the past 25 years -- not because mathematics is a beautiful art form, which it is -- but because mathematics is useful, which it also is. But this would seem to imply that a mathematician should convey some awareness of the main ways in which his subject is used.

On the other hand, a mathematics course cannot really include very much numerical analysis. Wilkinson's treatise [15] on computing eigenvalues is 700 pages long, and can hardly be summarized in every course on linear algebra! As a practical matter, then, the mathematics instructor's main responsibility is to be aware of the main features of practical computing in the areas of his mathematics courses, and mention occasional points of contact, while giving his students important references to important algorithmic materials in other books.

If one just ignores the relations between mathematics and its important applications, I fear that an instructor is running the risk of being exposed by some technological chapter of the Students for Democratic Society for not being relevant, and that is a very nasty accusation nowadays. Why risk it?

References

[1] R. Fletcher and M. J. D. Powell, "A rapidly convergent descent method for minimization", Computer J. 6 (1963), pp. 163-168.

[2] George E. Forsythe, "What is a satisfactory quadratic equation solver", pp. 53-61 of B. Dejon and P. Henrici (Editors), Constructive Aspects of the Fundamental Theorem of Algebra, Wiley-Interscience, 1969.

[3] George E. Forsythe and Cleve B. Moler, Computer Solution of Linear Algebraic Systems, Prentice-Hall, 1967.

[4] George E. Forsythe and Wolfgang R. Wasow, Finite-Difference Methods for Partial Differential Equations, Wiley, 1960.

[5] C. W. Gear, "The automatic integration of stiff ordinary differential equations", pp. A81-A85 of Anonymous, Proceedings IFIP Congress 68, North Holland Publishing Co., 1968.

[6] D. Hilbert, "Ein Betrag zur Theorie des Legendre'schen Polynoms", Acta Math. 18 (1894), pp. 155-160.

[7] William Edmund Milne, Numerical Solution of Differential Equations, Wiley, 1953.

[8] Cleve B. Moler, "Numerical solution of matrix problems", pp. 15-26 of Anonymous, The Digest Record of the 1969 Joint Conference on Mathematical and Computer Aids to Design, I.E.E.E. Catalogue No. 69 C 63-C, 1969.

[9] David E. Muller, "A method for solving algebraic equations using an automatic computer", Math. Tables and Other Aids to Computation 10 (1956), pp. 208-215.

[10] John von Neumann and H. H. Goldstine, "Numerical inverting of matrices of high order", Bull. Amer. Math. Soc. 53 (1947), pp. 1021-1099, and Proc. Amer. Math. Soc. 2 (1951), pp. 188-202.

[11] Michael R. Osborne, "A new method for the integration of stiff systems of ordinary differential equations", pp. A86-A90 of Anonymous, Proceedings IFIP Congress 68, North Holland Publishing Co., 1968.

[12] Anthony Ralston and Herbert S. Wilf, Mathematical Methods for Digital Computers, Wiley, Vol. 1, 1960, and Vol. 2, 1967.

[13] Irene A. Stegun and Milton Abramowitz, "Pitfalls in computation", J. Soc. Indust. Appl. Math. 4 (1956), pp. 207-219.

[14] J. H. Wilkinson, Rounding Errors in Algebraic Processes, Prentice-Hall, 1963.

[15] J. H. Wilkinson, The Algebraic Eigenvalue Problem, Oxford, Clarendon Press, 1965.

Unclassified
Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) Computer Science Department Stanford University Stanford, California 94305	2a. REPORT SECURITY CLASSIFICATION Unclassified 2b. GROUP --

3. REPORT TITLE

PITFALLS IN COMPUTATION, OR WHY A MATH BOOK ISN'T ENOUGH

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)

Manuscript for Publication (Technical Report)

5. AUTHOR(S) (First name, middle initial, last name)

George E. Forsythe

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
January, 1970	43	15

8a. CONTRACT OR GRANT NO. N00014-67-A-0112-0029 b. PROJECT NO. NR 044-211 c. d.	9a. ORIGINATOR'S REPORT NUMBER(S) CS-147 9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report) none

10. DISTRIBUTION STATEMENT

Releasable without limitations on dissemination.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
---	Office of Naval Research

13. ABSTRACT

The floating-point number system is contrasted with the real numbers. The author then illustrates the variety of computational pitfalls a person can fall into who merely translates information gained from pure mathematics courses into computer programs. Examples include summing a Taylor series, solving a quadratic equation, solving linear algebraic systems, solving ordinary and partial differential equations, and finding polynomial zeros. It is concluded that mathematics courses should be taught with a greater awareness of automatic computation. ( )

Unclassified
Security Classification

14 KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
errors						
computers						
numerical analysis						
pitfalls in computation						
floating-point numbers						
round off						