THE PERFIDIOUS POLYNOMIAL

James H. Wilkinson

1. INTRODUCTION

The problem of finding the roots of polynomial equations has played a key role in the history of mathematics and indeed our very concept of numbers has been steadily broadened by consideration of it. The part it has played might be summarized as follows.

Starting with the positive integers which man, with uncharacteristic generosity, ascribes to the Almighty, negative integers were introduced so that the equation $x + p = 0$ has a solution. The requirement that $px - q = 0$ should have a solution then led to the introduction of the field of rationals q/p. This provided a very rich number system but, although it has the impressive property that there are an infinite number of rationals between any two of them, consideration of the equation $x^2 = 2$ showed it to be inadequate. There are no integers p and q such that $p^2/q^2 = 2$. This led to the concept of irrational numbers, though it must be admitted that there was still a long way to go before the concept of the real continuum was put on a firm basis. Finally the equation $x^2 + 1 = 0$ led to the introduction of the 'number' i such that $i^2 = -1$ and, thence, to the concept of complex numbers $a + ib$.

There then came what is perhaps the most momentous 'discovery' in the history of mathematics. With the field of complex numbers we are at the end of this particular line of development. There is no need to extend our number system further in order that polynomial equations of higher degree should have solutions. The Fundamental Theorem of Algebra asserts that every polynomial equation with complex coefficients has a complex root. All early proofs of this theorem made some appeal to intuition though a proof by Gauss is particularly convincing. It depends on the assumption that if a closed curve encircling the origin one or more times is shrunk continuously to a point it must at some stage pass through the origin. Attempts to put the intuitive element in these proofs on a firm basis provided a valuable stimulus to mathematicians.

In parallel with these developments there were attempts to find explicit expressions for the roots of polynomial equations. For linear and quadratic equations this was trivial and the problem was solved for the cubic and quartic by the sixteenth century. Related results for the quintic proved to be elusive and, finally, it was shown by Abel and Galois that it is impossible, in general, to express the solution of equations of the fifth degree and higher in terms of radicals. Again work in this area yielded particularly rich dividends for mathematics in general.

In classical analysis too, polynomial functions have for long occupied a key position. In many respects they may be regarded as the simplest functions having any real character; they can be differentiated and integrated an arbitrary number of times. In spite of their simplicity they have the remarkable property expressed in the approximation theorem of Weierstrass—"Any continuous function may be approximated to any desired accuracy by a polynomial of sufficiently high degree."

The history of mathematics has, in a sense, contrived to make mathematicians feel 'at home' with polynomials. This is well illustrated by Weierstrass' theorem. Why should one be interested in the fact that continuous functions can be satisfactorily approximated by polynomials? The assumption is implicit in the theorem that this is a desirable thing to do since polynomials are such agreeable functions.

2. NUMERICAL EVALUATION OF POLYNOMIALS

The cosy relationship that mathematicians enjoyed with polynomials suffered a severe setback in the early fifties when electronic computers came into general use. Speaking for myself I regard it as the most traumatic experience in my career as a numerical analyst.

The early electronic computers had no hardware facilities for computation in floating-point arithmetic and operation in this mode was effected by means of subroutines. Having constructed such a set of subroutines for our first electronic computer, PILOT ACE, I looked for a simple problem on which to give them a field trial. Computing the largest real root of a real polynomial equation seemed tailor-made for the exercise. Moreover since polynomials of even quite a modest degree can assume values covering a very wide range over the interval including their zeros, the use of floating-point arithmetic is particularly appropriate. I was well aware that there were practical difficulties associated with the problem but imagined that they were mainly confined to equations with multiple roots or pathologically close roots. I felt that it was not the time to concern myself with these difficulties and I decided, therefore, to make two programs with quite limited objectives.

The first took n prescribed real numbers $z_1 > z_2 > \cdots > z_n$ and constructed the *explicit* polynomial $\prod(x - z_i)$ by multiplying the factors together. This in itself provided a simple test of the basic subroutines.

The second took the resulting polynomial $f(x)$ and, starting with the initial approximation $z_1 + 1$, used the Newton Raphson iterative method

$$x_{r+1} = x_r - f(x_r)/f'(x_r) \tag{2.1}$$

to locate the largest root.

It was my intention to avoid difficulties by choosing well separated z_i. An attractive feature of this problem is that the background theory is trivial. With exact computation the approximations tend *monotonically* to the root z_1 and convergence is ultimately quadratic.

After using these programs with complete success on examples of order three, I decided to give them their first genuine test. I took

$n = 20$ and the polynomial with roots $20, 19, \ldots, 2, 1$ and submitted it to the Newton Raphson program expecting immediate success. To my disappointment the iteration continued indefinitely and no root emerged. Since the polynomial was clearly devoid of all guile, I assumed that there was an error either in the floating-point subroutines or the two polynomial programs.

Failing to find an error in the subroutines and verifying that the computed polynomial was correct to working accuracy, I returned to the zero finder and examined the successive iterates x_r and the computed values $f(x_r)$. The x_r appeared to be virtually random numbers in the neighbourhood of 20 while the computed values were of arbitrary sign and were in any case much larger than the true values. (Notice that the true values of $f(x_r)$ could be computed directly via $\Pi(x_r - z_i)$ since we were in the privileged position of knowing the roots *a priori*.)

Now the interesting thing about this behaviour is that it would not have been difficult to predict, but it was so unexpected that it was some time before I could bring myself to attempt the analysis; I was convinced that either one of the programs or the computer was at fault.

3. ELEMENTARY ERROR ANALYSIS

We wish to avoid any detailed rounding error analysis in this article. Fortunately a very simplified analysis is quite adequate to demonstrate that the observed behaviour is to be expected. Let us concentrate on the evaluation of the polynomial

$$f(z) = z^n + a_{n-1}z^{n-1} + \cdots + a_1 z + a_0 \tag{3.1}$$

at $z = x$. This was achieved via the algorithm

$$s_n = 1$$

$$s_p = x s_{p+1} + a_p \ (p = n-1, \ldots, 1, 0), \tag{3.2}$$

$$f(x) = s_0,$$

usually known as *nested multiplication*. We note in passing that $z^{n-1} + s_{n-1}z^{n-2} + \cdots + s_2 z + s_1$ is the quotient polynomial and s_0 is the remainder when $f(z)$ is divided by $(z - x)$.

Consider now a hypothetical computation in which every operation is performed *exactly* except the addition involved in the computation of s_k. We assume that an error ε is made at this stage. This error could be the result of a 'blunder' or a mere rounding error. The computed $\bar{s}_i$ satisfy *exactly* the relations

$$\bar{s}_n = 1,$$

$$\bar{s}_p = x\bar{s}_{p+1} + a_p \qquad (p \neq k), \tag{3.3}$$

$$\bar{s}_k = x\bar{s}_{k+1} + a_k + \varepsilon.$$

(Of course the $\bar{s}_i$ will be the same as the s_i until we reach $\bar{s}_k$; after that they will differ.) Clearly

$$\bar{f}(x) = f(x) + \varepsilon x^k. \tag{3.4}$$

This is an exact relation and is in no sense dependent on ε being small. The performance is clearly dependent on the relative sizes of $|f(x)|$ and $|\varepsilon x^k|$. It will appear later that when x is in the neighbourhood of 20, the most dangerous term, in the case when ε arises from a rounding error, is s_{15}, so we concentrate on the case $k = 15$.

Suppose $x = 20 + y$ where $|y| \ll 1$, then

$$f(x) = y(1+y)(2+y)\cdots(19+y) \doteq 19!y \doteq 1.22 \times 10^{17} y \tag{3.5}$$

while

$$\varepsilon x^{15} \doteq \varepsilon(20)^{15} \doteq 3 \times 10^{19}\varepsilon. \tag{3.6}$$

We now have to decide the order of magnitude of ε. This error arises when we attempt to compute

$$x\bar{s}_{16} + a_{15} \tag{3.7}$$

and the coefficient a_{15} in our polynomial lies between 10^9 and 10^{10}. In floating-point arithmetic on the PILOT ACE the mantissa had 30 binary digits, roughly the equivalent of 9 decimal digits. Hence, since a_{15} requires ten digits for its representation, the rounding error involved in its mere representation may be as large as 5 units! If $|x\bar{s}_{16}| < |a_{15}|$ (as, in fact, it is), the rounding error made in this addition is also of that order. Taking ε to be 5 then, we have

$$\varepsilon x^{15} \doteqdot 1.5 \times 10^{20}. \tag{3.8}$$

If y really *is* appreciably less than unity, then from (3.5) and (3.8) $|f(x)|$ is far smaller than $|\varepsilon x^{15}|$. Even when $y \doteqdot 0.1$ the error is some 10^4 times as large as the true value. The computed value of $f(x)$ is completely dominated by this single rounding error. Since it could be positive or negative we would expect the computed values to be of random sign and far larger in value than the true values. In general there will be rounding errors in the computation of each s_p and on a nine-digit decimal computer no accuracy can be achieved in a computed value for any x in the range of, say, 12 to 22. It may readily be verified that when $y = 10^{-4}$, seventeen decimals are required in the computation in order to guarantee that even the sign of $\bar{f}(x)$ is correct. This is a very disappointing result and is all the more surprising since twenty is quite a modest degree; for a polynomial equation of, say, degree 100 with a similar distribution of roots the computational requirements are prohibitive. However the implications of the trivial error analysis we have just given are even more severe, as we shall show in later sections.

4. BACKWARD ERROR ANALYSIS

Let us return now to the equations defining the s_p and consider the true computing process with rounding errors made in every arithmetic operation. The computed values $\bar{s}_p$ satisfy the relations

$$\begin{aligned} &\bar{s}_n = 1, \\ &\bar{s}_p = x\bar{s}_{p+1} + (a_p + \varepsilon_{1p} + \varepsilon_{2p}), \\ &\bar{s}_0 = \bar{f}(x), \end{aligned} \tag{4.1}$$

where ε_{1p} is the rounding error made in the multiplication and ε_{2p} is the error made in the addition. Equations (4.1) are *exact* relations; in practice we shall not know ε_{1p} and ε_{2p} but we shall have upper bounds for them in terms of the computed precision, $|x\bar{s}_{p+1}|$ and $|a_p|$. We have bracketed a_p with ε_{1p} and ε_{2p} in order to highlight a simple property of the $\bar{s}_p$. Equations (4.1) state that the computed sequence is an *exact* sequence corresponding to a polynomial with coefficients $a_p + \varepsilon_{1p} + \varepsilon_{2p}$ for precisely the relevant argument. Although the observation is almost trivial, it is of profound practical importance. It shows that all the errors made in the computation are precisely equivalent in their effect to perturbing the original coefficients by $\varepsilon_{1p} + \varepsilon_{2p}$ and then performing the computation exactly. This is one of the simplest examples of a *backward error analysis*; it is so named since it reflects back all the rounding errors made in the course of the computation as equivalent errors in the data. Although basically a simple concept, many are uneasy when first introduced to it. There is a tendency to worry about the effect of errors in one $\bar{s}_p$ on the computation of subsequent values. From the point of view of a backward rounding error analysis this is irrelevant; such an analysis makes no comment on the errors in the $\bar{s}_p$ though it may be used subsequently to obtain bounds for them. Once we have admitted the perturbations in the data then the computed values are *exact*. There are several advantages in this form of analysis.

(i) It puts the rounding errors made during the computation on the same footing as the original errors in the data. In practice the data will seldom be exact so that one must of necessity consider the effect of errors in them.

(ii) It reduces the problem of determining the errors in the solution to that of the effect of perturbations in the data. Perturbation theory has been extensively studied in classical mathematics and hence there is a rich body of experience on which to draw.

(iii) The third advantage may be described as follows. Suppose we have a problem with data elements c_i and we can show that the computed solution is exact for data $c_i(1 + \varepsilon_i)$ and give bounds for $|\varepsilon_i|$. The size of these bounds for the $|\varepsilon_i|$

provides a basis for assessing the performance of the algorithm in the presence of rounding errors.

In connexion with (iii), the following considerations are of particular relevance. In any algorithm all of the data elements c_i must be used at least once, i.e., they must take part in some arithmetic operation. In general that operation will involve a rounding error and the effect of that error will be that we have effectively used $c_i(1+\eta_i)$ rather than c_i where $|\eta_i|$ is bounded by the relative precision ε of the basic computer operations. Hence it is virtually impossible for any practical algorithm to do better *in general* than to give the exact results for data $c_i(1+\varepsilon_i)$ where the $|\varepsilon_i|$ can take values up to ε. In so far as each c_i and its successors take part in a large number of operations, even a very stable algorithm is likely to give much larger values of ε_i than ε.

Now the remarkable thing about nested multiplication is that in spite of the fact that it can provide such poor computed values of $f(x)$ it is extremely stable in the sense we have described and is particularly so on precisely the example we discussed in section 2. In fact we shall show that each $\bar{f}(x)$ is exact for a polynomial with coefficients $a_i(1+\varepsilon_i)$ with bounds on $|\varepsilon_i|$ only marginally greater than the computer precision. Let us assume for the moment that $|a_p| > |x\bar{s}_{p+1}|$ really is true. It is not our intention in this paper to indulge in any detailed rounding error analysis since this tends to be indigestible. Instead we shall make our point by illustrating what happens on a six-decimal digit computer. Let us assume that the exponent of a_p is k and the exponent of $x\bar{s}_{p+1}$ is, say, $k-3$. Then we may write

$$a_p = 10^k(.u_1u_2u_3u_4u_5u_6) \tag{4.2}$$

$$x\bar{s}_{p+1} = 10^k(.000\,v_1v_2v_3v_4v_5v_6|v_7v_8v_9v_{10}v_{11}v_{12}) \tag{4.3}$$

where the u_i denote the six decimal digits in a_p. The exact value of $x\bar{s}_{p+1}$ (note that in a backward error analysis we are not concerned with the fact that $\bar{s}_{p+1}$ is in error) is a twelve-digit number with an exponent three less (by our assumption); hence when aligned with

a_p it starts with three zeros. The v_i represent the twelve digits. In practice the product $x\bar{s}_{p+1}$ will be rounded, giving an error bounded by $\frac{1}{2}$ in the digital position of v_6. The addition then takes place and the sum is rounded. The rounding at this stage is bounded by $\frac{1}{2}$ in the digital position of u_6, or just possibly one digit lower if cancellation takes place (N.B., one digital position at most can be lost by cancellation). Hence the bound for $|\varepsilon_{p1}|+|\varepsilon_{p2}|$ is only marginally greater than $\frac{1}{2}$ in the last digit of the computer representation of a_p. This must be regarded as a best possible result.

It is not difficult to justify the assumption that $|x\bar{s}_{p+1}| < |a_p|$ for an x in the region of interest. This is because (with exact computation) the s_p are the coefficients of the quotient polynomial when $f(z)$ is divided by $z-x$.

Our conclusion is that there is no alternative algorithm which is likely to give better results than nested multiplication since a relative perturbation of up to $\frac{1}{2}$ in the last digit of each a_i is virtually inevitable.

In Table 1 we give the computed values of $f(z)=(z-1)(z-2)\cdots(z-12)$ for a range of arguments in the neighbourhood of 10. These were obtained using floating-point computation with a mantissa of 46 binary digits. The polynomial of order 12 rather than 20 was used because some of the coefficients of the latter have too many digits to be represented exactly on a 46 binary digit computer.

Notice that the computed values of $f(z)$ for values of z from $10+2^{-42}$ to $10+2^{-23}$ are all roughly the same size and of unpre-

TABLE 1

z	Computed $f(z)$		True $f(z)$	
$10+1\times2^{-42}$	$+0.63811$		0.16501×10^{-6}	
$10+2\times2^{-42}$	$+0.57126$	Dominated	0.33003×10^{-6}	
$10+3\times2^{-42}$	-0.31649	by	0.49505×10^{-6}	Almost linear
$10+4\times2^{-42}$	-0.45823	rounding	0.66007×10^{-6}	over whole of
$10+2^{-28}+7\times2^{-42}$	$+0.29389$	errors	0.27048×10^{-2}	this interval
$10+2^{-23}+7\times2^{-42}$	$+0.70396$		0.86518×10^{-1}	
$10+2^{-18}+7\times2^{-42}$	$+0.33456\times10^{1}$		0.27685×10^{1}	
$10+2^{-13}+7\times2^{-42}$	$+0.89316\times10^{2}$		0.88608×10^{2}	

dictable sign. The contribution made by the rounding errors swamps the true value. It is not until we reach $z=10+2^{-13}$ that we are sufficiently far from the root at 10 for the value of $f(x)$ itself to dominate the rounding errors. The size of the computed values is perfectly predictable. The most dangerous term turns out to be a_8z^8. In fact, $a_8=749463$ and hence the contribution made by the rounding error when we compute s_8 is bounded by

$$2^{-46}\times749643\times10^8 \text{ (approximately)} \doteqdot \frac{7.50\times10^{13}}{2^{46}} \doteqdot \frac{7.50\times10^{13}}{7.04\times10^{13}} \doteqdot 1.00. \tag{4.4}$$

Accordingly, we can expect the rounding errors to contribute something of the order of magnitude of unity to the computed value of $\bar{f}(z)$ and it is therefore only when the true $f(z)$ is substantially larger than this that we have any correct significant figures. The table deserves careful study. Since 10 requires 4 binary digits for its representation, 2^{-42} is the value of the least significant digit on a 46-digit computer for any number in the neighbourhood of 10. Evaluation at $x=10$ gives no rounding errors so the *computed* value is equal to the true value, i.e., zero. Accordingly the first four arguments were taken to be the four successive computer numbers following 10 itself. For these $f(z)$ is of the order of 10^{-6} and this is completely swamped. Next we chose arguments which were essentially $10+2^{-28}$, $10+2^{-23}$, $10+2^{-18}$, $10+2^{-13}$ but in order to avoid the danger of atypical roundings each of these was augmented by 7×2^{-42}. With the third of these arguments $f(z)$ is just large enough to rise above the level of the rounding errors and $\bar{f}(z)$ has one correct decimal digit; with the fourth argument $f(z)$ has reached 10^2 and the computed $\bar{f}(z)$ now has rather better than two decimals of accuracy. It may be verified that for all eight values of the argument the error is of order unity with the sort of variation one might expect from the random fluctuations. Had we attempted the exercise with the polynomial $(x-1)(x-2)\cdots(x-20)$ on the same computer, the rounding errors would have dominated the computed value over most of the range $z+12$ to $z=22$.

5. SENSITIVITY OF THE ROOTS

So far we have concentrated on what might be regarded as the rather unexciting problem of evaluating $f(z)$, though we are primarily concerned with finding the roots of $f(z)=0$. However, if changes in the least digit in the coefficients a_i completely change the exact value of the polynomial for a given argument, it is obvious that the implications for the roots must be very serious. At this point we turn to perturbation theory and ask ourselves "What is the effect on the roots of a perturbation of a_i by εa_i?"

Let us denote the roots of the original polynomial by x_i. If ε is sufficiently small the root x_j becomes $x_j+\delta x_j$ where

$$|\delta x_j| \doteqdot |\varepsilon a_i x_j^i / f'(x_j)|$$
$$\doteqdot \left| \varepsilon a_i x_j^i \Big/ \prod_{i\neq j}(x_i-x_j) \right|. \tag{5.1}$$

For the polynomial we have discussed it can be verified that the greatest sensitivity is that of the root 15 (which is x_6) with respect to a perturbation in a_{15}. For this we have

$$|\delta x_6| = |\varepsilon a_{15}\cdot15^{15}/5!14!|, \tag{5.2}$$

and since $a_{15}\doteqdot1.67\times10^9$, this gives

$$|\delta x_6| \doteqdot 0.7\varepsilon\times10^{14}. \tag{5.3}$$

Even with ε as small as 10^{-14} the perturbation is 0.7 and this is so large that the first order theory is obviously inadequate.

The implications of this are quite calamitous. All coefficients of our polynomial from a_{15} to a_0 require more than 9 decimal digits for their representation. Hence on a nine-digit computer even the best possible representations of each of these coefficients will involve a rounding error and this will correspond to a value of ε between $\pm\frac{1}{2}10^{-9}$. In fact the PILOT ACE program for computing the polynomial performed extremely well; whenever all the given values x_i were of the same sign the computed coefficients were, in general, correct to working accuracy. Unfortunately, the exact roots of the computed polynomial differed wildly from the given x_i. We illustrate this in Table 2 by giving the roots of the polynomial equation $(z-1)(z-2)\cdots(z-20)-2^{-23}z^{19}=0$, which differs from the original polynomial only in the coefficient of z^{19}. The true a_{19} is 210, which requires 8 binary digits for its representation; on a 30-binary digit computer the perturbation 2^{-23} therefore represents a unit in the first position beyond the end of the computer representation of a_{19}. It will be seen from the table that the smaller roots are scarcely affected but by the time we reach the root $x=7$ the perturbation has become substantial, while the ten roots from 10 to 19 have become five complex conjugate pairs; the roots 18 and 19, for example, have become $19.5\cdots \pm(1.9\cdots)i$. Perturbations in a_{15} and a_{16} are even more devastating in their effect.

6. IMPLICATIONS OF THE ERROR ANALYSIS

We now turn to the implications of the results of sections 2 to 5 for practical numerical analysts. Polynomial equations arise in very many branches of 'applied mathematics', using that term in its widest sense. However, these equations do not present themselves directly in explicit polynomial form. Perhaps the most important illustration of this is provided by the algebraic eigenvalue problem which arises in connexion with vibration problems in physics, chemistry and engineering. Here the polynomial equation is in the determinantal form

$$\det(\lambda I - A) = 0, \tag{6.1}$$

TABLE 2
Roots of $(z-1)(z-2)\cdots(z-20)-2^{-23}2^{19}=0$ correct to 9 decimal places

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

which for an $n \times n$ matrix gives a monic polynomial equation of degree n; however, the primary data are the elements of A. Now it was perfectly natural for mathematicians to approach this problem by devising algorithms which would give coefficients of the *explicit* polynomial corresponding to $\det(\lambda I - A)$, i.e., the characteristic polynomial of A. This had the advantage of reducing the volume of data from the n^2 elements of A to the n elements of the characteristic polynomial. However, the real incentive to the development of such algorithms was that it reduced the problem to the solution of an explicit polynomial equation, and this was felt to be a highly desirable transformation. Although attempts were made to analyse the effect of the rounding errors made in the algorithm on the accuracy of the computed coefficients of the explicit polynomial form, the desirability of this form does not seem to have been questioned. Almost all of the algorithms developed before the 1950s for dealing with the unsymmetric eigenvalue problem (i.e., problems in which A is unsymmetric) were based on some device for computing the explicit polynomial equations.

So far we have based our exposition of the numerical problems associated with computing the roots of polynomial equations on a specific polynomial of degree twenty. This polynomial now enjoys a certain notoriety; it is sometimes referred to as 'Wilkinson's remarkable polynomial' and the opinion is quite widely held that there is something unusual about it. This is a particularly disappointing development. The polynomial was first used by me quite by chance; it was in fact the first polynomial of order 20 that came to mind; so, far from trying to invent a particularly intractable polynomial, I used it precisely because I imagined (quite wrongly) that it would be free from any numerical difficulties. It has the advantage that the orders of magnitude of $f(x)$ and $f'(x)$ may easily be determined in terms of factorials.

The really disturbing fact is that it is quite typical of polynomials with real coefficients and real roots and, indeed, many polynomials which arise in practice behave much worse than this. As an experiment I once took some 100 eigenvalue problems of order 25 for which I had found the eigenvalues by an algorithm that is known to be very stable with respect to eigenvalues. Knowing the eigenvalues, I then computed the minimum accuracy to which the explicit polynomial would need to be computed in order to ensure that it would give the eigenvalues correct to 5 decimals. (This would be a reasonable requirement in practice.) In every single case it would have been necessary to obtain the coefficients correct to more than 20 decimals and for many of the examples a much higher accuracy would have been necessary.

Again it should be emphasized that the polynomial $(z-1)(z-2)\cdots(z-20)$ is not a 'difficult' polynomial *per se*. I have sometimes been asked what precision of computation was necessary in order to determine the roots in Table 2 to 9 decimal places. This question is asked in the mistaken belief that this must be a difficult numerical problem. The 'difficulty' with the polynomial $\Pi(z-i)$ is that of evaluating the *explicit* polynomial accurately. If one already knows the roots, then the polynomial can be evaluated without any loss of accuracy. The roots in Table 2 were computed using a zero finder and evaluating the function directly by means of the expression $\Pi(z-i)-2^{-23}z^{19}$. Eleven decimal computation was perfectly adequate to determine the roots quoted.

Practical computer users sometimes find it difficult to believe that there can be any justification for determining the coefficients of an explicit polynomial to high accuracy when the primary data are known to much lower accuracy. The fallacy in this argument may be exposed by quite a simple example. Consider the determinantal equation

$$\det\begin{bmatrix} 1+\varepsilon_1-\lambda & \varepsilon_3 \\ \varepsilon_4 & 1+\varepsilon_2-\lambda \end{bmatrix}=0. \tag{6.2}$$

The explicit polynomial form is

$$\lambda^2-(2+\varepsilon_1+\varepsilon_2)\lambda+(1+\varepsilon_1)(1+\varepsilon_2)-\varepsilon_3\varepsilon_4=0 \tag{6.3}$$

and the roots are

$$\frac{(2+\varepsilon_1+\varepsilon_2)\pm\left((\varepsilon_1-\varepsilon_2)^2+4\varepsilon_3\varepsilon_4\right)^{1/2}}{2}. \tag{6.4}$$

Observe that under the square root we have only second-order terms in the ε_i, the potential first-order terms cancel out. If the ε_i are of the order of 10^{-6}, the two roots differ from unity by quantities of order 10^{-6}. Random perturbations of order 10^{-6} in the ε_i make changes of order 10^{-6} in these roots. However, random independent perturbations of order 10^{-6} in the coefficients of the explicit polynomial make perturbations of order 10^{-3} in the roots. Perturbations in the ε_i in (6.2) produce perturbations in the explicit polynomial (6.3) which are so correlated as to make their effect on the roots far less important. Unfortunately, rounding errors made in the coefficients are inevitably uncorrelated in general.

Determinantal equations of the type $\det(A-\lambda I)=0$ are commonly such that the roots are relatively insensitive to independent perturbations in the a_{ij}. This may be true even when the roots include close clusters. In fact when A is symmetric the roots are well-conditioned in this respect, however sensitive the roots of the corresponding explicit polynomial equation may be. The numerical instability introduced in transforming to the explicit polynomial equation is an induced instability.

7. AMELIORATION OF ILL-CONDITIONING

The sensitivity of the roots of explicit polynomial equations is something which is inherent in that mode of representation. All algorithms which involve rounding errors are, in general, doomed to give results of disappointing accuracy relative to the computer precision when used to deal with an ill-conditioned polynomial.

It is natural to ask whether the ill-condition can be removed by some simple transformation of the variables. The answer is that sometimes it can but the transformation itself must be performed to high accuracy because, as we have remarked, the transformed polynomial will be exactly related to a polynomial $\Sigma(a_i+\varepsilon_i)z^i$ with ε_i at best of the order of the precision used.

An interesting example in this respect is provided by the polynomial equation

$$z^3-3.0000002z^2+2.9999997z-1.0000004=0. \tag{7.1}$$

This polynomial is very close to $(z-1)^3$ and, hence, the roots can be expected to be very sensitive to random perturbations in the coefficients such as will occur in the evaluation of the polynomial. It is attractive to make the transformation $z-1=w$ and thus to obtain a cubic equation in w which no longer has close roots. Now the standard algorithm for determining the coefficients of the transformed polynomial uses repeated division by $z-1$, the successive remainders providing the coefficients. As we have remarked, the process of dividing by $z-\alpha$ is precisely the same as evaluating the polynomial at $z=\alpha$. Now if the transformation is performed using standard 8-decimal digit, floating-point arithmetic the resulting polynomial equation is

$$w^3-2\times10^{-7}w^2-7\times10^{-7}w-9\times10^{-7}=0, \tag{7.2}$$

and this provides accurate roots. This would seem to contradict the claim that the transformation must be performed in higher precision but that is an illusion. The fact is that no rounding errors occur in this reduction, so we may regard it as having been done to arbitrarily high precision even though we appeared to be using only 8-decimal digit computation. Had we been foolish enough to make, say, the transformation $w=z-1.0000123$, then rounding errors would have occurred and these would have been just as damaging as those arising in the use of the original polynomial.

Representation of (7.1) in the form

$$(z-1)^3=10^{-7}(2z^2+3z+4) \tag{7.3}$$

also enables one to compute the roots accurately via the iterative procedure

$$(z_{r+1}-1)^3=10^{-7}(2z_r^2+3z_r+4), \tag{7.4}$$

provided care is taken to select the same branch throughout for iteration to a specific root. Again this is possible because the transformation from (7.1) to (7.3) has been performed *exactly*.

8. MULTIPLE ROOTS

The equation $(z-1)(z-2)\cdots(z-20)=0$ was adopted initially precisely because it has 'well separated' roots. The objective was to avoid the known difficulties associated with multiple roots or root clusters. Since the numerical problems proved to be unexpectedly severe, it is pertinent to ask whether the selected polynomial equation can be thought of as in any sense related to one with multiple roots.

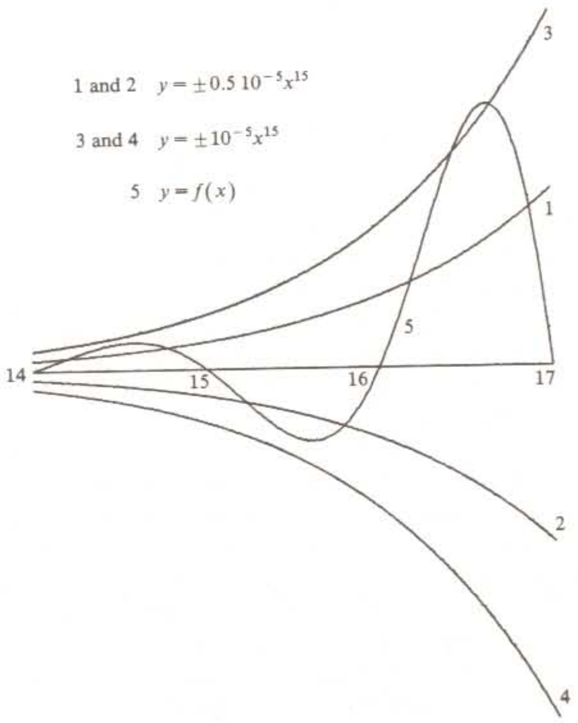
In fact very small relative perturbations in the coefficients can give an equation with multiple roots. To see this consider, for example, the perturbed equation

$$(x-1)(x-2)\cdots(x-20)-\varepsilon x^{15}=0. \tag{8.1}$$

The roots of this equation occur at the points of intersection of the curves $y=f(x)\equiv(x-1)(x-2)\cdots(x-20)=0$ and $y=g(x)\equiv+\varepsilon x^{15}$. The first curve is symmetric about the point $x=10.5$, crosses the axis at $x=i$ $(i=1,\ldots,20)$ and has peaks in between these values, alternatively positive and negative, at the zeros of the derivative. Consider now values of ε of order 10^{-5}. For values of x from, say, 0 to 6, $g(x)$ is *very* close to the x-axis compared with the peaks of $f(x)$, and the points of intersection of $f(x)$ with $g(x)$ are virtually the same as the points of intersection of $f(x)$ with the x-axis. Hence (as we have already seen) the early roots are scarcely affected. However, as x increases, $g(x)$ increases quite rapidly while $f(x)$ continues to dissipate its energies by oscillating in sign. When we reach values of x, say, from 14 to 17, $g(x)$ is fully comparable with $|f(x)|$ even at the peaks of the latter. This is illustrated in Fig. 1 for $\varepsilon=\pm(0.5)10^{-5}$ and $\varepsilon=\pm(1.0)10^{-5}$. For $\varepsilon=+(0.5)10^{-5}$ the roots 14 and 15 have moved towards each other to a substantial extent as have also the roots 16 and 17. For $\varepsilon=(1.0)10^{-5}$ the roots originally at 14 and 15 have become complex conjugate pairs and the roots originally at 16 and 17 have almost coalesced. For an ε between these two values, $y=g(x)$ will just touch $y=f(x)$, giving a double root between 14 and 15. A slightly larger value of ε will produce a double root between 16 and 17. Similarly, for negative ε, a value between $-(0.5)10^{-5}$ and

$-(1.0)10^{-5}$ will give a double root between 15 and 16. Since $a_{15} \doteq (1.67)10^9$ these ε represent relative perturbations of the order of 10^{-14}. It is an instructive exercise to determine perturbations which give two pairs of double roots and other combinations. A more difficult question is the following.

Suppose we have a computer with an inadequate word length for the exact representation of the polynomial $(x-1)(x-2)\cdots(x-20)$, for example, a ten-digit decimal floating-point computer. The correctly rounded version of the polynomial equation has roots which are very different from the true values. Determine the ten-digit decimal polynomial such that its roots z_i give the minimum value of $\Sigma|z_i - i|$ (or any other appropriate measure of the difference between the two sets of roots).



$y=(x-1)(x-2)\cdots(x-20)$ and $y=k(10^{-5}x^{15})$
for $x=14$ to 17

FIG. 1.

9. WELL-CONDITIONED POLYNOMIALS

Although it is important to realize that explicit polynomial equations with ill-conditioned roots are remarkably common, not all polynomials of high degree are like that. The polynomial equations

$$x^n \pm 1 = 0 \tag{9.1}$$

are extremely well-conditioned for all values of n. A perturbation ε in any coefficient gives a perturbation of order ε/n in all the roots.

Of particular interest for the following section are polynomial equations with roots $\alpha, \alpha^2, \ldots, \alpha^n$. If we take $\alpha = 1/2$, $n = 20$, we find that the coefficients vary enormously in order of magnitude. In fact, in terms of orders of magnitude, the polynomial is

$$\begin{aligned} &x^{20} + x^{19} + 2^{-1}x^{18} + 2^{-4}x^{17} + 2^{-8}x^{16} + 2^{-13}x^{15} + 2^{-19}x^{14} \\ &+ 2^{-26}x^{13} + 2^{-34}x^{12} + 2^{-43}x^{11} + 2^{-53}x^{10} + 2^{-64}x^{9} \\ &+ 2^{-76}x^{8} + 2^{-89}x^{7} + 2^{-103}x^{6} + 2^{-118}x^{5} + 2^{-134}x^{4} \\ &+ 2^{-151}x^{3} + 2^{-169}x^{2} + 2^{-189}x + 2^{-210} \end{aligned} \tag{9.2}$$

At first sight this appears to be an unsatisfactory state of affairs; moreover there is a natural tendency to regard the roots as dangerously close with a cluster near zero. Such fears prove to be completely unfounded. Small relative perturbations in all coefficients make correspondingly small *relative* perturbations in each of the roots. Even the smallest root behaves well. Since rounding errors made in evaluating a polynomial are always equivalent in their effect to small relative perturbations in the coefficients, one can predict that this polynomial equation will behave well. In fact even the rather unsophisticated program used on the PILOT ACE was capable of determining any one of the roots almost to full working accuracy.

To establish this result we observe that for a perturbation εa_k in a_k, the relative change in root x_i is

$$|\delta x_i/x_i| \sim \varepsilon |a_k| 2^{-ki}/(2^{-i}|PQ|), \tag{9.3}$$

where

$$P = (2^{-i}-2^{-1})(2^{-i}-2^{-2})\cdots(2^{-i}-2^{-i+1})$$
$$Q = (2^{-i}-2^{-i-1})(2^{-i}-2^{-i-2})\cdots(2^{-i}-2^{-20}). \tag{9.4}$$

We may write

$$|P| = 2^{-\frac{1}{2}i(i-1)}[(1-2^{-1})(1-2^{-2})\cdots(1-2^{-i+1})]$$
$$|Q| = 2^{-i(20-i)}[(1-2^{-1})(1-2^{-2})\cdots(1-2^{-20+i})] \tag{9.5}$$

and the bracketed expressions are both convergents to the infinite product

$$(1-2^{-1})(1-2^{-2})(1-2^{-3})\cdots. \tag{9.6}$$

All convergents certainly lie between $1/2$ and $1/4$ and hence

$$|\delta x_i/x_i| < \varepsilon|a_k|2^{s+4} \quad \text{where } s = \tfrac{1}{2}i[41-2k-i]. \tag{9.7}$$

For a fixed value of k this takes its maximum where $i = 20-k$, so that

$$|\delta x_i/x_i| < \varepsilon|a_k|2^{4+\frac{1}{2}(20-k)(21-k)}. \tag{9.8}$$

However from (9.2) it may be verified that all a_k satisfy

$$|a_k| \leqslant 4.2^{-\frac{1}{2}(20-k)(21-k)} \tag{9.9}$$

and hence

$$|\delta x_i/x_i| < 2^6\cdot\varepsilon. \tag{9.10}$$

A relative perturbation of ε in any coefficient can be amplified at worst by the factor 2^6 though for most i and k this is an overestimate since the above inequalities are quite crude. This polynomial then is well-behaved and for smaller values of α its behaviour is even better. However, such polynomials proved to have a sting in their tails.

10. POLYNOMIAL DEFLATION

The Fundamental Theorem of Algebra asserts that every polynomial equation over the complex field has a root. It is almost beneath the dignity of such a majestic theorem to mention that in fact it has precisely n roots. The latter result is regarded as trivial and is usually mentioned only as an aside. When a root x_1 has been found, the polynomial $f(z)$ may be divided by $z-x_1$ to give a quotient polynomial of degree $n-1$

$$f(z) = (z-x_1)q(z). \tag{10.1}$$

By the Fundamental Theorem $q(z)$ has a root and the process can be continued to give the n roots.

I must have been subconsciously influenced by this attitude because when I turned to polynomial deflation (as the above process is called), I was not expecting substantial difficulties. Since I knew by this time that the polynomial with roots 2^{-i} was well-conditioned, I used this polynomial as my first test. The largest root 2^{-1} was found first and this had an error of only 1 in its least significant digit. The approximation to the root 2^{-2} derived from the deflated polynomial was of disappointing accuracy and later roots proved to have virtually no relation to the correct roots even as regards their orders of magnitude.

Again backward error analysis is an invaluable tool for demonstrating the cause of this difficulty. Suppose we have accepted a value $x_1+\varepsilon$ as an approximation to x_1. Let us consider the result of performing the deflation process exactly using this approximation. We have

$$f(z) = (z-(x_1+\varepsilon))q(z)+r \tag{10.2}$$

where $q(z)$ is the deflated polynomial and r is the remainder. In practice, of course, r is ignored and work is continued with $q(z)$. Now, by our assumption, equation (10.2) is exact and hence

$$(z-(x_1+\varepsilon))q(z)=f(z)-r \text{ (exactly)}. \qquad (10.3)$$

This equation implies that $x_1+\varepsilon$ and the exact roots of $q(z)$ are those of $f(z)-r$, not of $f(z)$ itself. However, from the Remainder Theorem, $r=f(x_1+\varepsilon)$ and since

$$f(z)=(z-x_1)(z-x_2)\cdots(z-x_n), \qquad (10.4)$$

we have

$$r=f(x_1+\varepsilon)=\varepsilon(x_1-x_2+\varepsilon)(x_1-x_3+\varepsilon)\cdots(x_1-x_n+\varepsilon). \qquad (10.5)$$

Consider now the implications of this for a polynomial equation with roots $x_i=2^{-i}$ when the accepted approximation to x_1 is $2^{-1}+10^{-10}$ and, therefore, has an error only in the tenth decimal. From (10.5)

$$r=\varepsilon(2^{-1}-2^{-2}+\varepsilon)(2^{-1}-2^{-3}+\varepsilon)\cdots(2^{-1}-2^{-20}+\varepsilon) \qquad (\varepsilon=10^{-10}). \quad (10.6)$$

Each of the terms in parenthesis is certainly greater than 2^{-2} and hence

$$r>\varepsilon 2^{-38}=10^{-10}2^{-38}>2^{-72}. \qquad (10.7)$$

With exact deflation the roots of $q(z)=0$ will be roots of $f(z)-r$, which has the constant term $2^{-210}-r$ instead of 2^{-210}. However, r is larger than 2^{-210} by a factor greater than $2^{138}\doteqdot 10^{41}$. Since the product of the roots of a monic polynomial equation is equal to the constant term and this has been changed by an enormous factor, the roots of $q(z)$ cannot possibly give good approximations to the remaining roots of $f(z)$. It is hardly likely that when rounding errors are made in the deflation process itself the results will be superior to those resulting from exact deflation with the approximate root.

On the other hand, suppose an approximation $2^{-20}+\varepsilon$ to 2^{-20} with a low relative error has been accepted. We have now

$$r=f(2^{-20}+\varepsilon)=(2^{-20}+\varepsilon-2^{-1})(2^{-20}+\varepsilon-2^{-2}) \cdots(2^{-20}+\varepsilon-2^{-19})\varepsilon \qquad (10.8)$$

and we assume that $\varepsilon=10^{-16}$, which again corresponds to a relative error of approximately one part in 10^{10}. We require an approximation to $|r|$ and, fortunately, quite a crude bound will suffice. We have certainly

$$|r|<2^{-1}\cdot 2^{-2}\cdots 2^{-19}\cdot 10^{-16}=2^{-190}\times 10^{-16}\doteqdot 2^{-210}\times 10^{-10}. \qquad (10.9)$$

Hence the constant term 2^{-210} is changed by less than one part in 10^{10}, and we know that this has little effect on any of the roots.

Although this is a very well-conditioned polynomial, we see that division by an approximate factor corresponding to the largest root is disastrous even if performed exactly; division by an approximate factor corresponding to the smallest root is completely harmless. This analysis is entirely born out in practice. A similar result is true in general for polynomials having roots of widely varying magnitudes.

The above argument suggests that roots should be found in terms of increasing absolute magnitude. Unfortunately, with many of the more efficient algorithms it is not easy to ensure that roots *are* found in this order. It is natural to ask whether there is some stratagem which overcomes this problem. In fact there is a very simple solution.

It is a well-established tradition when dividing a polynomial of degree n by an (exact) linear factor to obtain the coefficient of z^{n-1} first and then those of $z^{n-2}, z^{n-3},\ldots$ in succession. With an exact factor and exact division there is, of course, no remainder. However, we could just as well start at the other end and find the constant term first, followed by those of $z, z^2,\ldots$ in succession;

again there will be no remainder. The quotient polynomials will, of course, be exactly equal.

If α is merely an approximate root, these quotient polynomials will not be exactly equal; the remainder will be a constant with the conventional mode of division and a multiple of z^n when the division is performed in the reverse direction. Now we may think of division in the reverse direction in the following terms. Suppose we write $z = w^{-1}$, then

$$f(z) = f(1/w) = \left(a_0 w^n + a_1 w^{n-1} + \cdots + a_n\right)/w^n = g(w)/w^n \tag{10.10}$$

$$(z-\alpha) = (1-\alpha w)/w = (\alpha^{-1} - w)\alpha/w \tag{10.11}$$

and if we write $\beta = \alpha^{-1}$, we can think of dividing $(w-\beta)$ into

$$\left(a_0 w^n + a_1 w^{n-1} + \cdots + a_n\right).$$

If α is the largest root of $f(z)$, then β is the smallest root of $g(w)$. If a large root of $f(z)$ has been found, then clearly it should be divided out in the reverse direction.

However, this still leaves us with the necessity of finding either the largest root or the smallest at each stage. If we find one of the inner ones, such as 2^{-10} in our example, then neither quotient is satisfactory. Suppose now we denote the two quotients by

$$b_{n-1}z^{n-1} + b_{n-2}z^{n-2} + \cdots + b_1 z + b_0, \tag{10.12}$$

and

$$c_{n-1}z^{n-1} + c_{n-2}z^{n-2} + \cdots + c_1 z + c_0, \tag{10.13}$$

respectively. We could use a composite quotient using coefficients $b_{n-1}, b_{n-2}, \dots, b_r$ and $c_{r-1}, c_{r-2}, \dots, c_0$. Note that in dividing forward, b_{n-1} is found first and in dividing backwards, c_0 is found first; hence it seems appropriate to use those terms which are computed earlier in the two processes. Will there always be a good choice of r, presumably different according to which root one happens to have found?

The answer to this question is a resounding 'yes'. However, we must be clear what we mean by this. There is a limitation on the accuracy with which any root can be found even when iterating in the original polynomial. This is a fundamental property of the polynomial with respect to that root. The irreducible error in a root is that corresponding to relative perturbations in the original coefficients which are a modest multiple of the computer precision. It would be unreasonable to expect that the deflation process has positive virtues that enable us to find ill-conditioned roots more accurately after several deflations than by iteration in the original polynomial, assuming a fixed precision of computation throughout. In fact, by using the appropriate value of r when each approximate root has been accepted, the accuracy obtained in each root is almost equal to that attainable by iteration for that root in the original polynomial.

It should be appreciated that if the original polynomial equation is very ill-conditioned, the successive deflated polynomials will usually become steadily better conditioned. When, for example, the roots $1, 2, \dots, 15$ of our notorious polynomial equation have been found, the remaining quintic with roots $16, \dots, 20$ is very much better conditioned even though this includes several of the worst conditioned roots of the original. This improvement in the condition avails us little. When the first approximate root has been accepted and the appropriate decomposition (with rounding errors) has been performed, the deflated polynomial will be related to a slightly perturbed version of the original. These perturbations, small as they are, will already have severely damaged ill-conditioned roots. Even if the remaining $n-1$ roots and deflations are performed exactly from that point onwards we can never recover from the loss of accuracy already inflicted.

However, there are circumstances in which this loss of accuracy may not be serious. This is when our primary objective is to produce a set of approximate roots $\bar{x}_i$ such that $\prod(z-\bar{x}_i)$ is 'close' to the given polynomial. A good iterative method for finding a single root plus the *composite* deflation we have described achieves just this. Though the errors in the ill-conditioned roots will be severe (except when the rounding errors happen, fortuitously, to be benign) the errors in the complete set will be correlated in such a way as to reproduce the original polynomial remarkably accurately. For a detailed explanation of this and the details of composite deflation the reader is referred to Peters and Wilkinson [7]. The error analysis involved is not particularly complex but its presentation here would violate the policy adopted in this article. However, a trivial example illustrates how this comes about.

Consider the polynomial equation

$$z^2 - 2z + 1 = 0, \tag{10.14}$$

which has $z = 1$ as a double root. This root is accordingly fairly ill-conditioned. If iteration is performed using six-decimal floating-point computation then, say, $x = 1.00024$ will be accepted as a root since the computed value of $f(x)$ will be exactly zero. Notice that the smallest attainable computed value (other than zero) is 10^{-6}, since the last step is to add xs_1 to a_0, i.e., to 1 to give s_0. The smallest value is attained when computed $xs_1 = -0.999999$. However, when deflation is performed (composite deflation is irrelevant here) the computed quotient is

$$z - 0.999760, \tag{10.15}$$

and the second computed root is 0.999760. Both roots have substantial errors but

$$(z - 1.00024)(z - 0.999760) = z^2 - 2z + 0.9999999424 \tag{10.16}$$

and the original polynomial is given to much greater accuracy than are the individual factors. Such a phenomenon is referred to as 'backwards stability' and is of great importance in numerical analysis.

11. CONCLUSIONS

The title of this article reflects my feelings in early encounters with solving polynomial equations but perhaps it is a misnomer. Polynomial equations may be said to have played an 'instructive' role in the history of mathematics and could have continued to play this role for numerical analysts in the period immediately following the advent of electronic computers.

There *was* an intense interest in the effect of rounding errors at that time but, unfortunately, it was focused primarily on matrix problems, particularly the solution of systems of linear equations. These problems had a superficial formal complexity which had the effect of making their analysis seem rather difficult. This formal complexity is completely absent from the problem of evaluating a polynomial or even of locating a simple real root of a real polynomial equation. Viewed in retrospect it is interesting that I used backward error analysis both in polynomial evaluation and in polynomial deflation almost, as it were, by accident. (The term backward error analysis had not been coined at that time and it was certainly not in use as a recognized tool.) Although I was astonished, indeed affronted, by my experiences with simple polynomials, I was rather pleased with the effectiveness of the analysis. Nevertheless it did not impress me sufficiently to make me adopt similar policies in the analysis of matrix problems. My conversion to backward error analysis in a matrix context did not occur until several years later. I had observed that for stable methods of solving the eigenvalue problem the residual vector $\boldsymbol{r}$ defined by

$$\boldsymbol{r} = Ax - \lambda x \tag{11.1}$$

was of the order of magnitude of the computer precision times $\|A\|$. Quite suddenly it occurred to me that this implied that

$$(A - \boldsymbol{r}x^T)x = \lambda x \tag{11.2}$$

(assuming x is normalized so that $x^Tx = 1$) and that λ and x were therefore exact for $A - \boldsymbol{r}x^T$, which is a 'very close neighbour' of A. This *did* indeed ultimately persuade me consciously to adopt backward error analysis as a working tool and it yielded rich dividends almost immediately. By this time Givens [3] had already used it quite specifically in his analysis of the reduction of a real symmetric matrix to tri-diagonal form by orthogonal similarities. It is interesting that this did not lead to its explicit adoption by others or indeed by Givens himself in the analysis of other algorithms.

For accidental historical reasons therefore backward error analysis is always introduced in connexion with matrix problems. In my opinion the ideas involved are much more readily absorbed if they are presented in connexion with polynomial equations. Perhaps the fairest comment would be that polynomial equations narrowly missed serving once again in their historical didactic role and rounding error analysis would have developed in a more satisfactory way if they had not.

REFERENCES

1. G. Birkhoff and S. Mac Lane, *A Survey of Modern Algebra*, 4th Edition, Macmillan, New York, 1977.
2. B. Dejon and P. Henrici (Editors), *Constructive Aspects of the Fundamental Theorem of Algebra*, Proceedings of a symposium at the IBM Research Laboratory, Zurich-Ruschlikon, June 1967, Interscience, London, New York, 1969.
3. J. W. Givens, "Numerical computation of the characteristic values of a real symmetric matrix," Oak Ridge National Laboratory, ORNL-1574, 1954.
4. A. S. Householder and F. L. Bauer, "On certain methods for expanding the characteristic polynomial," *Numer. Math.*, **1** (1959), pp. 29–39.
5. M. A. Jenkins and J. F. Traub, "A three-stage variable shift iteration for polynomial zeros and its relation to generalized Rayleigh iteration," *Numer. Math.*, **14** (1970), pp. 252–263.
6. F. W. J. Olver, "Evaluation of zeros of high degree polynomials," *Phil. Trans. Roy. Soc.*, **244** (1952), pp. 385–415.
7. G. Peters and J. H. Wilkinson, "Practical problems arising in the solution of polynomial equations," *J. Inst. Math. Appl.*, **8** (1971), pp. 16–35.
8. G. W. Stewart, "Some iterations for factoring a polynomial," *Numer. Math.*, **13** (1969), pp. 458–471.
9. J. H. Wilkinson, "The evaluation of zeros of ill-conditioned polynomials," Part I, *Numer. Math.*, **1** (1959), pp. 150–166, Part II, *Numer. Math.*, **1** (1959), pp. 167–180.
10. _____, "Error analysis of floating point computation," *Numer. Math.*, **2** (1960), pp. 319–340.
11. _____, *Rounding Errors in Algebraic Processes, Notes on Applied Sciences No.* 32, Her Majesty's Stationery Office, London, 1963.
12. _____, *The Algebraic Eigenvalue Problem*, Oxford University Press, London, 1965.

NEWTON'S METHOD

Jorge J. Moré and D. C. Sorensen

1. INTRODUCTION

Many fundamental problems in science, engineering, and economics can be phrased in terms of minimizing a scalar valued function of several variables. Problems that arise in these practical settings usually have constraints placed upon the variables. Special techniques are required to handle these constraints but eventually the numerical techniques used must rely upon the efficient solution of unconstrained minimization problems.

Newton's method plays a central role in the development of numerical techniques for optimization. One of the reasons for its importance is that it arises very naturally from considering a Taylor approximation to the function. Because of its simplicity and wide applicability, Newton's method remains an important tool for solving many optimization problems. In fact, most of the current practical methods for optimization (e.g., quasi-Newton methods) can be viewed as variations on Newton's method. It is therefore important to understand Newton's method as an algorithm in its own right and as a key introduction to the most recent ideas in this area.