The Mathematical Basis and a Prototype Implementation of a New Polynomial Rootfinder with Quadratic Convergence

T. E. HULL and R. MATHON
University of Toronto

Formulas developed originally by Weierstrass have been used since the 1960s by many others for the simultaneous determination of all the roots of a polynomial. Convergence to simple roots is quadratic, but individual approximations to a multiple root converge only linearly. However, it is shown here that the mean of such individual approximations converges quadratically to that root. This result, along with some detail about the behavior of such approximations in the neighborhood of the multiple root, suggests a new approach to the design of polynomial rootfinders. It should also be noted that the technique is well suited to take advantage of a parallel environment. This article first provides the relevant mathematical results: a short derivation of the formulas, convergence proofs, an indication of the behavior near a multiple root, and some error bounds. It then provides the outline of an algorithm based on these results, along with some graphical and numerical results to illustrate the major theoretical points. Finally, a new program based on this algorithm, but with a more efficient way of choosing starting values, is described and then compared with corresponding programs from IMSL and NAG with good results. This program is available from Mathon (combin@cs.utoronto.ca).

Categories and Subject Descriptors: G.1.0 [**Numerical Analysis**]: General—*error analysis*; *numerical algorithms*; G.1.5 [**Numerical Analysis**]: Roots of Nonlinear Equations—*convergence*; *error analysis*; *iterative methods*; *methods for polynomials*; G.4 [**Mathematics of Computing**]: Mathematical Software—*algorithm analysis*

General Terms: Algorithms, Design

Additional Key Words and Phrases: Polynomial rootfinder, polynomial zerofinder, quadratic convergence

1. INTRODUCTION

Suppose that

$$P(z) = c_n z^n + c_{n-1} z^{n-1} + \cdots + c_0,$$

This work was supported by the Natural Sciences and Engineering Research Council of Canada and the Information Technology Research Centre of Ontario.
Authors' address: Department of Computer Science, University of Toronto, Toronto, Ontario, Canada M5S 3G4; email: {tehull; combin}@cs.utoronto.ca.

where $c_0, c_1, \ldots, c_n$ are in general complex coefficients, and $c_n \neq 0$.

Suppose also that $z_1, z_2, \ldots, z_n$, where the z_i are distinct, are approximations to the zeros of $P(z)$. Then consider the following iteration formulas:

$$\bar{z}_j = z_j - P(z_j)/Q'(z_j), \quad 1 \leq j \leq n,$$

where

$$Q(z) = c_n(z - z_1)(z - z_2)\cdots(z - z_n).$$

Since the 1960s these formulas have been used by many persons as the basis of methods for calculating approximations to all the roots of a polynomial simultaneously. The formulas were introduced much earlier, by Weierstrass [1891], but in another context related to the fundamental theorem of algebra. Much later Dochev [1962] developed these formulas for the solution of polynomial equations. Using a different approach, they were rediscovered by Kerner [1966]. (Durand [1960] had taken an approach like Kerner's, but did not carry it to the point of developing formulas in the closed form shown here.) We will refer to the formulas as the WDK formulas.

We have experimented with these formulas for a number of years, using starting values distributed around a circle which is known to contain all the roots of the polynomial and using stopping criteria similar to what was proposed by Adams [1967] and Peters and Wilkinson [1971]. In earlier work with Trevizan [1977] and Solowiej [1983], as well as in more recent comparisons we have made with root-finding programs in the IMSL [1985] and NAG [1991] libraries, we have found programs based on these formulas to be both accurate and reliable—but relatively very slow.

The main purpose of this article is to develop ways in which root-finding programs based on these formulas can be speeded up. The main new mathematical result is that the *average* of the m approximations to a root of multiplicity m converges quadratically to that root, whereas the individual approximations converge only linearly. We show how one can take advantage of this result, and we give a specific example of how this idea can be used to provide a considerable improvement over using the WDK formulas alone. The convergence is much faster, especially in the presence of multiple or near-multiple roots, since the convergence there is quadratic. Error bounds can also be calculated, and tests show that the results are reliable and very accurate.

However, convergence is very slow at the start, so it is necessary to incorporate a faster way of getting starting values for the method. We modify a conventional Newton process to provide rough approximations to the roots as starting values, and the resulting program is tested very carefully, with good results. The program is available, but should be considered only a prototype at this stage. We plan to develop a production code in the near future based on this prototype.

In Section 2 we give for completeness a simple derivation of the WDK formulas. Convergence to simple roots is established in Section 3, and to multiple roots in Section 4. Some detail about behavior near a multiple root is discussed in Section 5, and error bounds are developed in Section 6.

An algorithm based on the results in Sections 2–6 is outlined in Section 7. The main idea is to monitor progress in WDK iterations by, at intermediate stages in the calculation, using the individual error bounds to identify clusters of approximations that are close together and determining if the average of any such cluster can be taken to be a root of corresponding multiplicity. In Section 8 a specific program based on the ideas outlined in Section 7 is used to provide graphical and numerical results to illustrate the behavior of this approach.

Finally, in Section 9, the faster start mentioned above is incorporated in a program called DROOTS, and this program is compared over a large number of test problems to corresponding programs from the IMSL [1985] and NAG [1991] libraries.

2. THE WDK FORMULAS

The polynomial in Section 1 can be rewritten in terms of its roots:

$$P(z) = c_n(z - r_1)(z - r_2)\cdots(z - r_n),$$

where the roots r_j are not necessarily distinct.

Recall that the approximations, $z_1, z_2, \ldots, z_n$, to these roots, where the z_j *are* distinct, were used to define

$$Q(z) = c_n(z - z_1)(z - z_2)\cdots(z - z_n).$$

Now, suppose new approximations, $\bar{z}_1, \bar{z}_2, \ldots, \bar{z}_n$, are introduced where $\bar{z}_j = z_j + \Delta z_j$, $1 \le j \le n$, and $R(z)$ is defined by

$$R(z) = c_n(z - \bar{z}_1)(z - \bar{z}_2)\cdots(z - \bar{z}_n).$$

Then we can expand $R(z)$ in powers of the Δz's to obtain

$$R(z) = Q(z) - c_n \sum_{k=1}^{n} \Delta z_k \prod_{\substack{j=1 \\ j \neq k}}^{n} (z - z_j) + p(z),$$

where $p(z)$ is a perturbation consisting of higher-order terms in the Δz's.

If we neglect $p(z)$ and insist that what is left be exactly $P(z_j)$ when $z = z_j$, we can solve for the Δz's to obtain

$$\Delta z_j = -P(z_j)/Q'(z_j),$$

so that

$$\bar{z}_j = z_j - P(z_j)/Q'(z_j),$$

which are the WDK formulas. This simple derivation of the formulas is essentially the one given by Aberth [1973].

An earlier derivation was based on the fact that the roots of $P(z)$ satisfy the following equations:

$$\sum_{j=1}^{n} r_j = -c_{n-1}/c_n,$$

$$\sum_{j=1}^{n} \sum_{k>j}^{n} r_j r_k = c_{n-2}/c_n,$$

$$\vdots$$

$$\prod_{j=1}^{n} r_j = (-1)^n c_0/c_n,$$

where the left sides are the elementary symmetric functions of the roots. Durand [1960] took this approach and proposed using Newton's method for solving this system of n nonlinear equations for all the r's. This leads to a system of n linear equations for our Δz's. Kerner [1966] followed this line of reasoning as well, but he also found analytic solutions for the system of linear equations, which turn out to be exactly the WDK formulas. (Dochev's [1962] approach was altogether different, but led to the same formulas.)

For later purposes (Section 4) it is worth noting that the first of the n linear equations for the Δz's is

$$\sum_{j=1}^{n} \Delta z_j = -\sum_{j=1}^{n} z_j - c_{n-1}/c_n,$$

so that the sum of the new approximations in each iteration, and consequently their average, will remain constant, and equal to $-c_{n-1}/c_n$, throughout the iteration process.

Two features of the WDK formulas should be noted. One is that, since the approximations are determined simultaneously, there is no need to be concerned about difficulties introduced by deflation as there is with more traditional methods. The other feature is that the formulas are obviously well suited to take advantage of a parallel environment. This latter aspect has been investigated by Freeman [1989] and Freeman and Bane [1991].

In a serial environment it is natural to consider a Gauss-Seidel modification, and there is some experimental evidence that this is helpful (e.g., see Milovanović and Petković [1986]); however, our further analysis will assume the Jacobi approach directly implied by the formulas, and some of our results would not be valid if the Gauss-Seidel modification is used.

3. CONVERGENCE TO SIMPLE ROOTS

From the WDK formulas we have immediately that, for $1 \le j \le n$,

$$\begin{aligned}\bar{z}_j - r_j &= z_j + \Delta z_j - r_j \\ &= z_j - r_j - (z_j - r_j)\prod_{\substack{k=1 \\ k\neq j}}^{n} \frac{z_j - r_k}{z_j - z_k}.\end{aligned}$$

Now suppose that $\epsilon = \max_{1\le j\le n}|z_j - r_j|$ and note that

$$\frac{z_j - r_k}{z_j - z_k} = 1 + \frac{z_k - r_k}{z_j - z_k}.$$

Then, if r_j is a simple root (so that, for sufficiently small ϵ, $|z_j - z_k|$ is bounded away from 0 for all $k \neq j$), we have

$$\frac{z_j - r_k}{z_j - z_k} = 1 + O(\epsilon),$$

so that

$$\begin{aligned}\bar{z}_j - r_j &= (z_j - r_j)(1 - (1 + O(\epsilon))) \\ &= O(\epsilon^2).\end{aligned}$$

Thus we have quadratic convergence to simple roots, as one would expect of a Newton method.

4. CONVERGENCE TO MULTIPLE ROOTS

Let us now consider convergence to a multiple root, r say, of multiplicity m. We can assume that $r_1 = r_2 = \cdots = r_m = r$, and that $r_{m+1}, r_{m+2}, \ldots, r_n$ are different from r, but not necessarily from each other. It is sufficient to consider this one case.

From the WDK formulas we now have, for $1 \le j \le m$,

$$\bar{z}_j - r = z_j - r - A\prod_{k=m+1}^{n}\left(\frac{z_j - r_k}{z_j - z_k}\right),$$

where

$$A = \frac{(z_j - r)^m}{\prod_{\substack{k=1 \\ k\neq j}}^{m}(z_j - z_k)}.$$

As before we suppose that $\epsilon = \max_{1 \le j \le n} | z_j - r_j |$. Then the terms of the product in the expression for $\bar{z}_j - r$ can be shown as before to be $1 + O(\epsilon)$, so that

$$\bar{z}_j - r = z_j - r - A(1 + O(\epsilon)).$$

If we now further assume that the method does converge at least linearly, so that $\bar{z}_j - r = O(\epsilon)$, we can conclude that $A = O(\epsilon)$, and

$$\bar{z}_j - r = z_j - r - A + O(\epsilon^2).$$

To show that the convergence of the mean is quadratic, we sum the above over all j for $1 \le j \le m$. But the sum of the terms on the right, apart from the $O(\epsilon^2)$ terms, is exactly the sum of the approximations one would get by applying the WDK formulas to the equation $w^m = 0$, where $w = z - r$. And we know from the Durand-Kerner approach mentioned in Section 2 that the WDK formulas will always produce approximations whose sum is correct, namely, 0 in this case. We can therefore conclude that

$$\sum_{j=1}^{m} \bar{z}_j \Big/ m - r = O(\epsilon^2),$$

as required.

To prove this quadratic convergence of the mean to multiple roots, we had to assume more than just that $\max_{1 \le j \le n} | z_j - r_j | = \epsilon$, for otherwise a term in the denominator of A could, for example, be $O(\epsilon^2)$, in which case A could increase indefinitely as $\epsilon \to 0$.

We assumed that the $\bar{z}$'s converged at least linearly. If we had not made this assumption we would have had to assume something to make sure the z's approximating r were kept reasonably separated, so that A could not increase indefinitely. For example, we could have assumed that $| z_j - z_k | \ge c\epsilon$ for some $c > 0$ and all $j \ne k$ with $1 \le j, k \le m$. A more restrictive, but quite plausible assumption that satisfies our needs, and which is a good indication of what happens in practice, will be discussed in the next section.

5. BEHAVIOR NEAR A MULTIPLE ROOT

We now examine the behavior of the calculated approximations to a multiple root in the neighborhood of that root. We suppose that r is a multiple root of multiplicity m, specifically that

$$P(z) = (z - r)^m F(z),$$

where $F(r) \ne 0$.

We can interpret what we did in Section 2 to determine the Δz's in the

WDK formulas was to require that

$$R(z) = P(z) + p(z),$$

for $z = \bar{z}_j$, $1 \le j \le m$. But $R(\bar{z}_j) = 0$, so that we have

$$(\bar{z}_j - r)^m F(\bar{z}_j) + p(\bar{z}_j) = 0,$$

and this means that the m $\bar{z}$'s which approximate r can be represented by

$$\bar{z}_j = r + \left(\frac{-p(\bar{z}_j)}{F(\bar{z}_j)} \right)^{1/m}.$$

This suggests that the approximations to a multiple root tend to approach that root from uniformly spaced positions around a circle with center at the multiple root—and this in turn suggests that the mean of those approximations could be a particularly good approximation to the root. This kind of behavior, or something like it, has been observed experimentally by a number of authors, such as Aberth [1973], Pasquini and Trigiante [1985], and Smith [1970], and will be illustrated later (Section 8). Related results about the behavior of approximations near a multiple root have been given by Wilkinson [1963, p.39].

Suppose we assume the approximations to a multiple root are distributed in such a manner. Specifically, let us assume that, for $1 \le j \le m$,

$$z_j = r + \delta e^{(2\pi j/m + \alpha)i}(1 + O(\epsilon)),$$

for some fixed δ and α. Then, proceeding as in Section 4, we obtain again

$$\bar{z}_j - r = z_j - r - A(1 + O(\epsilon)),$$

where now

$$A = (z_j - r) \prod_{\substack{k=1 \\ k \ne j}}^{m} \frac{z_j - r}{z_j - z_k}$$

$$= (z_j - r) \prod_{\substack{k=1 \\ k \ne j}}^{m} \frac{\delta e^{(2\pi j/m + \alpha)i}}{\delta e^{(2\pi j/m + \alpha)i} - \delta e^{(2\pi k/m + \alpha)i}}(1 + O(\epsilon))$$

$$= (z_j - r) \prod_{\substack{k=1 \\ k \ne j}}^{m} \frac{1}{1 - e^{(2\pi(k-j)/m)i}}(1 + O(\epsilon))$$

$$= (z_j - r) \prod_{k=1}^{m-1} \frac{1}{1 - e^{(2\pi k/m)i}} (1 + O(\epsilon))$$

$$= (z_j - r) \frac{1}{m} (1 + O(\epsilon)).$$

This last equality follows from the fact that

$$\prod_{k=1}^{m-1} (1 - e^{(2\pi k/m)i}) = \lim_{z \to 1} \left(\frac{z^m - 1}{z - 1} \right) = m.$$

Thus, we have, for $1 \leq j \leq m$,

$$\bar{z}_j - r = (z_j - r) \left(1 - \frac{1}{m} (1 + O(\epsilon)) \right) (1 + O(\epsilon))$$

$$= \frac{m - 1}{m} (z_j - r) + O(\epsilon^2).$$

This shows that our new assumption about the individual approximations to a multiple root is preserved, with δ replaced by $(m - 1)\delta/m$, which is not surprising for a Newton method near a root of multiplicity m. It also establishes the *linear* convergence of the individual approximations. (For the special case $m = 1$, we of course have quadratic convergence.)

Consider now the mean of the approximations to r. We have

$$\bar{z}_j - r = \frac{m - 1}{m} (\delta e^{(2\pi j/m + \alpha)i}) + O(\epsilon^2),$$

so that the mean

$$\left(\sum_{j=1}^{m} \bar{z}_j - r \right) \Big/ m = \frac{m - 1}{m^2} \delta e^{\alpha i} \sum_{j=1}^{m} e^{(2\pi j/m)i} + O(\epsilon^2)$$

$$= O(\epsilon^2),$$

since the sum on the right is the sum of the mth roots of unity, which is 0. This establishes the quadratic convergence of the mean, under the new assumption about the distribution of the z's.

6. ERROR BOUNDS

We want now to determine error bounds, first for an individual approximation to a root and then for the mean of a cluster where the mean is being considered as the approximation to a multiple root.

From the theory of partial fractions we can write

$$P(z)/Q(z) = 1 - \sum_{k=1}^{n} \Delta z_k \Big/ (z - z_k),$$

provided of course that $P(z_k) \neq 0$. If we put $z = r_j$, we have

$$1 = \sum_{k=1}^{n} \Delta z_k \Big/ \left(r_j - z_k \right).$$

Taking absolute values, we obtain

$$1 \leq \sum_{k=1}^{n} |\Delta z_k \Big/ (r_j - z_k)|.$$

The maximum term in the sum will occur at a particular value of k, and for that value of k,

$$1 \leq n|\Delta z_k/(r_j - z_k)|,$$

so that

$$|z_k - r_j| \leq n|\Delta z_k|.$$

The right side is the first bound we wished to obtain. We thus have a bound associated with each approximation. (This does not mean that there is a root within each bound of the corresponding approximation—only that there is an approximation within its bound of each root. However, the union of the discs enclosed by approximations and their associated bounds does contain all the roots—so the bounds are valid in the Gerschgorin sense. It can also be shown by the usual continuation argument that, if the discs are disjoint, each disc contains exactly one root (e.g., see Aberth [1988, p.79]).)

Another bound can be obtained by first going back to

$$1 \leq \sum_{k=1}^{n} |\Delta z_k \Big/ (r_j - z_k)|$$

and noticing that if k is such that $| r_j - z_k |$ is a minimum for $1 \leq k \leq n$, then, for that value of k,

$$|z_k - r_j| \leq \sum_{k=1}^{n} |\Delta z_k|.$$

We have a choice between the two bounds we have just found, or we could use whichever is the smaller of the two.

We turn now to finding an error bound for the mean of a cluster. In this case we assume that the polynomial $Q(z)$ approximating $P(z)$ has roots z_k of multiplicity m_k and that $P(z_k) \neq 0$, $1 \leq k \leq s$. We can now write

$$P(z)/Q(z) = 1 - \sum_{k=1}^{s} \sum_{\ell=1}^{m_k} d_{k\ell} \Big/ (z - z_k)^{\ell},$$

where

$$d_{k\ell} = \frac{1}{(m_k - \ell)!} \left[P(z) / \prod_{\substack{i=1 \\ i \neq k}}^{s} (z - z_i)^{m_i} \right]^{(m_k - \ell)}_{z = z_k}.$$

If we put $z = r_j$, we can conclude that

$$1 \leq \sum_{k=1}^{s} \sum_{\ell=1}^{m_k} |d_{k\ell}/(r_j - z_k)^{\ell}|.$$

The maximum term in the sum will occur at particular values of k and ℓ, and, for those values,

$$1 \leq n |d_{k\ell}/(r_j - z_k)^{\ell}|.$$

This yields the bound

$$|z_k - r_j| \leq \max_{1 \leq \ell \leq m_k} (n |d_{k\ell})^{1/\ell}.$$

We now have an error bound for the mean of a cluster. (If the jth disc is disjoint from the rest, then it contains exactly m_j roots of $P(z)$ (see Aberth [1988, p.83]).)

It was indicated earlier in Section 1 that starting values could be obtained by distributing them around a circle which is known to contain all the roots of the polynomial. One way of choosing such a circle is to consider all the roots to be just one cluster, take its center to be the mean of all the roots, i.e., $-c_{n-1}/(nc_n)$, and to take the radius to be the bound just found in

```
ROOTFINDER(n : int, coeffs : complex 0..n, ind: int,
        roots: complex 1..n, errors : real 1..n)
    look after special cases (e.g., n = 1, 2)
    determine starting values (the z's)
    other initializations (numits, maxits, etc.)
    loop
        use WDK formulas to compute Δz's
        compute individual bounds and use them to
            identify clusters and determine centers of clusters
        if convergence or numits = maxits then
            set ind accordingly
    ..... exit loop
        end if
        update z := z + Δz
        numits := numits + 1
    end loop
    compute cluster bounds
    assign to roots and errors
    return
end ROOTFINDER
```

Fig. 1. Outline of an algorithm which shows how the formulas and ideas of Sections 2–6 can be organized into a procedure for finding simultaneous approximations to all the roots of a polynomial.

the special case when it is assumed that $Q(z)$ has only one root, say z_0, of multiplicity n, at the mean. This leads to

$$\max_{1\le\ell\le n}\left(\frac{n}{(n-\ell)!}\left|\frac{P^{(n-\ell)}(z_0)}{c_n}\right|\right)^{1/\ell}$$

as the radius.

7. AN ALGORITHM TO ILLUSTRATE

We present the outline of an algorithm in Figure 1. It shows how the formulas and ideas in the preceding sections can be organized in a relatively straightforward way to determine approximations to the roots of a polynomial. In the next section we will show how a program based on this outline demonstrates the main ideas in the preceding sections, and we indicate that such a program can be both reliable and accurate. Making such a program more efficient and more generally useful will be described in Section 9.

We first discuss the outline in more detail. The algorithm expects the input of n (the degree of the polynomial) and *coeffs* (an array of the $n + 1$ complex coefficients of the polynomial). It returns *roots* (an array of n approximations to the complex roots of the polynomial), *errors* (an array of

n real error bounds associated with the roots), and *ind* (an integer value to indicate why the return occurred, because convergence was achieved, or the maximum number of allowed iterations was reached, or some error condition occurred).

The procedure first looks after special cases, such as when $n = 1$ or 2, and possibly 3 or 4, or when there are leading or trailing zero coefficients. It then determines starting values—for example, by distributing the initial approximations around a circle which is known to contain all the roots, using known bounds, or the bound at the end of the preceding section. (In Section 9 we obtain starting values in a more efficient way.) Then other initializations are needed, such as setting the iteration counter *numits* to zero and assigning a maximum number of iterations to *maxits*.

The first statement and the last two statements inside the loop carry out the basic WDK iteration, and they keep track of how often the iteration has been completed. The other statements inside the loop start with a situation in which the z's and Δz's are known. This situation is analyzed, first by using the Δz's to determine individual error bounds according to one or both of the first two bounds given in Section 6. These bounds, along with their associated z's define n discs in the complex plane. Isolated discs correspond to simple roots. The others form clusters, which are identified, and the mean of each cluster is determined.

To determine if convergence has occurred, the program uses a running error analysis (e.g., as in Adams [1967] or Peters and Wilkinson [1971]) to determine if, for each cluster of multiplicity m, the error bounds in evaluating the polynomial and its first $m - 1$ derivatives are greater than the corresponding calculated values of the polynomial and its derivatives. If they are in each case, convergence is assumed. (By including the situations in which $m = 1$ in the above test, we have also included the simple roots.)

If convergence has occurred, or if the number of iterations has reached the maximum allowed, the program sets the indicator *ind* to an appropriate value, and then assigns appropriate values to the *roots* and *errors* arrays. It already has all these values available, except that it must compute the bounds for the clusters with $m > 1$, and to do that it uses the remaining formulas for the bounds given in Section 6.

Error returns might also occur. For example, overflow or underflow might occur, but most programming languages make it difficult or even impossible to cope with such situations in a convenient way. Sometimes such situations can be anticipated, in which case an error return can be made, along with an appropriate value of *ind*.

8. A PRELIMINARY PROGRAM AND SOME ILLUSTRATIVE RESULTS

For a preliminary test of the ideas outlined in Section 7 we have developed a double-precision Fortran subroutine for finding roots of polynomials, along with a test program which determines the speed and accuracy of

polynomial rootfinders over a large number of test problems. The programs were run on a Sun 4/40 in Fortran 77 (compiler version 1.4).

This rootfinder assumes that $n \leq 200$. It does not make any provision for special cases. For starting values, it first calculates a center and radius as described at the end of Section 6. It distributes $n - 1$ starting values around the circle so that the radii through these values make the angles $2\pi k/n$, $1 \leq k \leq n - 1$, with a horizontal line through the center, and then chooses a final starting value so that its angle is π/n. The maximum number of iterations is arbitrarily set to 200, and the bounds for determining clusters are obtained from the minimum of the first two bounds given in Section 6. In this way a disc is associated with each approximation. A collection of approximate roots is identified as a cluster if the union of their discs form a continuous region which is disjoint from all the other discs.

The test for convergence is based on the running error analysis described by Peters and Wilkinson [1971], but using Kahan's suggestion for reducing the number of operations [Peters and Wilkinson 1971].

Our test program has two special features. If the roots of a test polynomial are not supplied, it can find those roots by calculating them first in very high precision and then rounding them to double precision. (The results are saved for future testing.) It can also determine a measure of accuracy of the approximations produced by any particular rootfinder. To do this it uses an algorithm for matching the true roots pairwise with the approximations in such a way that the maximum deviation of the true values, from the approximations with which they are paired, is a minimum.

The first test problem is the following:

Problem 8.1. A monic polynomial with

—a simple root at (1, 2),

—a triple root at (3, −1), and

—a quintuple root at (5, 3).

Graphical output from the test program (Figure 2(a)) shows the progression of approximations produced by our rootfinder. Figure 2(b) gives a closer view of what happens near the quintuple root, and Figure 2(c) shows a closeup of what happens near the quintuple root. The main point that is illustrated here is that the program has analyzed clusters near the multiple roots, and at one stage, long before the individual approximations get very near those roots, it finds that the means of the individual approximations pass the convergence test for multiple roots. It then accepts these means as multiple roots, calculates their error bounds, and terminates execution. By contrast, Figure 2(d) shows what happens near the quintuple root if the test for multiplicity is removed from the rootfinder.

It should also be noted that the individual approximations to the quintuple root converge on that root in the "equiangular" manner suggested in Section 5. We have found this to be the case in all our test results.

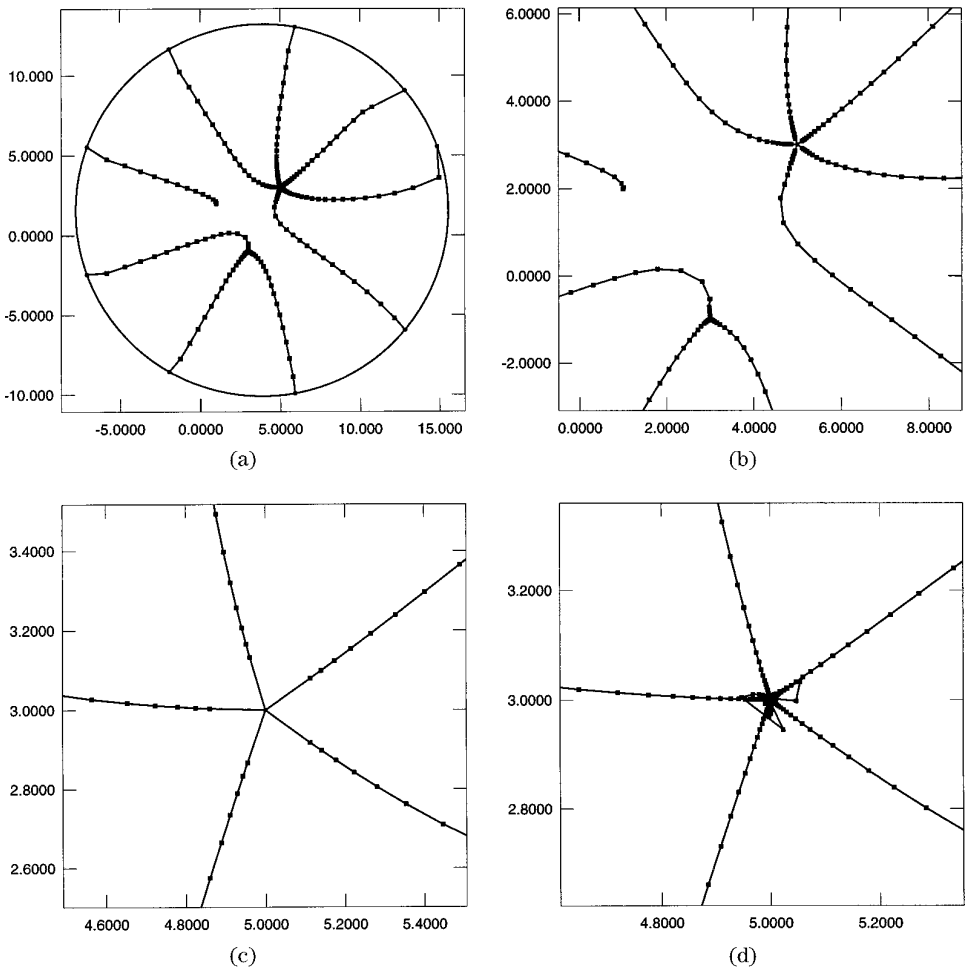

Fig. 2. (a) Overall view of progress of rootfinder on Problem 8.1; (b) closer view of progress toward quintuple root at (5,3); (c) closeup of progress near the quintuple root at (5,3); (d) closeup if test for multiplicity is removed from rootfinder.

The test program determined that the true error for each of the three roots was bounded by the error bound produced by the rootfinder. These errors were: true errors of 2.7×10^{-15}, 1.8×10^{-11}, and 0.46×10^{-9}, and computed bounds of 7.8×10^{-15}, 5.3×10^{-11}, and 2.4×10^{-9}, for the roots (1, 2), (3, -1), and (5, 3), respectively. When the test for multiplicity was removed the errors in the approximations near (5, 3) were approximately 10^{-2}, despite having used twice as many iterations (48 versus 24).

The results for this test problem are typical of what happened with a large number of test problems, including all of the test problems suggested by Jenkins and Traub [1975]. Figure 3(a) gives an overall view of what happens when our rootfinder is presented with one like the last (and very challenging) problems suggested by Jenkins and Traub. This is the following problem:

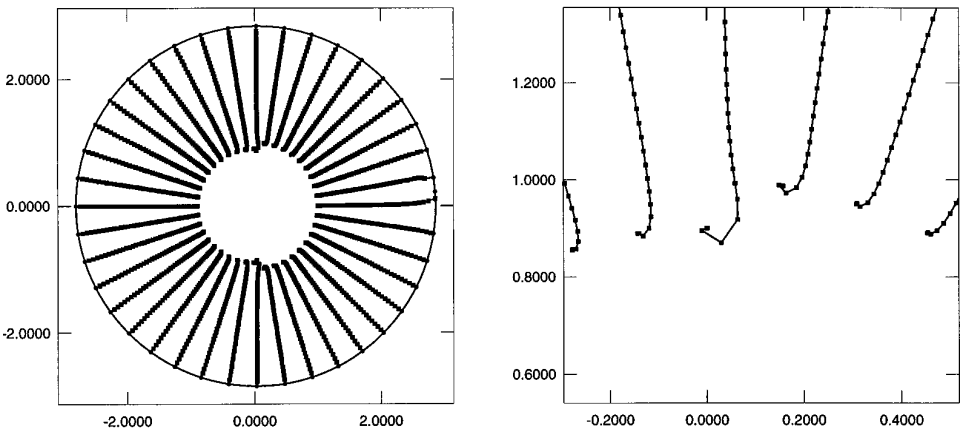

Fig. 3. (a) Overall view of progress of rootfinder on Problem 8.2; (b) closer view of progress toward a few of the roots of Problem 8.2 near (0,1).

Problem 8.2. The monic polynomial based on

—roots at $e^{k\pi i/20}$, $k = -9, -8, \ldots, 9$ and at
—$0.9\ e^{k\pi i/20}$, $k = 10, 11, \ldots, 30$.

(By "based on" we mean we used these values to determine the coefficients of the polynomial we work with, the roots of which are then found by using very high precision. Because of rounding errors these roots will differ from the original roots on which the polynomial is based.) Figure 3(b) gives a closer view of what happens for a few of the roots of this polynomial near (0, 1).

The true error for each of the 40 approximations was less than the corresponding computed error bound. The maximum of these true errors was 1.57×10^{-16}, while the maximum of the computed error bounds was 1.85×10^{-15}. There are no multiplicities to contend with here. The point is only to illustrate that the rootfinder can handle some quite challenging polynomials.

9. AN EFFICIENT IMPLEMENTATION

Up until now we have been primarily interested in the mathematical properties of the WDK formulas and the related ideas described in Sections 2–6 and in testing their behavior, especially in the neighborhood of multiple roots. To speed up the program used in the preceding section, we introduce an alternative starting procedure based on a conventional Newton method. It tries to find the starting values in increasing order of magnitude, deflating in the appropriate way described by Peters and Wilkinson [1971] as each value is accepted. It also arbitrarily restricts the number of iterations per starting value to 50. The other main modification we made to the program in the preceding section was to prevent the method

from producing approximations outside the bounding circle. Any such approximation is arbitrarily moved to the point on the circle where the line drawn from the approximation to the center of the circle meets the circumference. The only other changes we made in the program are that we made the stopping criterion slightly more stringent than what was used in the preceding section, and we removed the provision for producing the graphical output of the preceding section. We call the resulting Fortran subroutine DROOTS.

We compared DROOTS with the corresponding IMSL [1985] and NAG [1991] subroutines and obtained the results shown in Tables I and II. All programs were run on a Sun 4/75 in Fortran 77 (compiler version 1.4). Time shown in the tables is in milliseconds, while *error* is the maximum error observed when the approximate roots are matched with the true roots in such a way as to minimize the maximum error.

Most of the polynomials in Table I are random polynomials, as indicated, but the final two polynomials P9 and P40 are those used in the preceding section to illustrate the behavior of the WDK formulas. The observed maximum errors are smaller for these last two polynomials than those given in the preceding section. It is not surprising that they are different, since the starting values have been obtained in a completely different way, and the convergence is quadratic so that one extra iteration can make a considerable difference. That they are smaller is mainly due to the fact that we also made the convergence criterion in DROOTS more stringent. Similar comparisons have been made with many other polynomials, and the results for 20 of these are shown in Table II.

From Table I we see that DROOTS is the most accurate for all but three of the random polynomials (R54, R109, and R201) and is especially accurate for P9 and P40. NAG is fastest except for P9 and P40. In Table II, DROOTS is most accurate for 12 of the 20 polynomials and is fastest for 9 of the 20 (although particularly slow on polynomials 8, 9, and 13).

DROOTS appears to be at least competitive with the other two programs. We have some evidence that DROOTS is also somewhat more reliable, but we do not yet have enough evidence of this to provide sound documentation. Two other advantages of DROOTS are that it produces error bounds as well as approximations to the roots, and as indicated earlier, it is well suited to take advantage of parallelism (see, for example, Freeman [1989] and Freeman and Bane [1991]).

It can be seen from the tables that, following the Newton start, the WDK iterations provide 7 or 8 extra decimal digits of accuracy in almost all of the random polynomials, and 11 in P9 and P40. The range is much more variable in Table II, from 0 to 15, but averaging 6. (In polynomials 8, 9, and 13 WDK iterations provide 11, 6, and 3 extra such digits respectively.)

In Table I, time spent in the start routine is approximately one third of the total time, whereas in Table II it is much more variable. This is especially true for polynomials 8, 9, and 13. Apparently the start routine does not provide very good starting values in these 3 cases.

Table I. Results Using DROOTS and Corresponding Programs from the IMSL and NAG Libraries, First on Random Polynomials of Degree 5 (R50–R59), 10 (R100–R109), and 20 (R200–R209), where a Polynomial of Degree $5t$ has t Random Zeros in a Square of Side $2 \times 10i$, for $i = -2, -1, 0, 1, 2$ Centered at the Origin, and then on the Two Polynomials Used in the Preceding Section P9 (Problem 8.1) and P40 (Problem 8.2) (Time is in Milliseconds, and Error is the Maximum Error Observed)

	DROOTS				IMSL		NAG	
P(z)	Total Time	Time in Start	Final Error	Error in Start	Total Time	Final Error	Total Time	Final Error
R50	5.1	2.2	8.88E−16	7.40E−08	5.0	1.78E−15	3.5	9.16E−16
R51	4.7	1.9	3.66E−15	1.93E−08	4.6	1.42E−14	3.8	1.42E−14
R52	4.7	1.9	1.14E−16	1.04E−07	4.6	3.55E−15	3.7	3.55E−15
R53	5.3	2.5	1.24E−16	1.01E−09	5.2	6.42E−15	3.7	1.42E−14
R54	4.3	2.1	1.42E−14	1.26E−07	4.6	7.11E−15	3.8	1.59E−14
R55	4.9	1.9	2.22E−16	2.14E−07	4.6	1.43E−14	4.0	1.43E−14
R56	5.1	2.1	8.88E−16	7.46E−08	4.9	3.55E−15	3.5	3.55E−15
R57	4.8	2.0	9.93E−16	2.47E−09	4.6	1.42E−14	3.8	1.42E−14
R58	5.1	2.1	8.89E−16	3.35E−08	4.7	2.66E−15	4.1	3.55E−15
R59	4.8	2.1	1.39E−17	6.41E−09	4.6	1.19E−16	3.5	1.99E−15
R100	17.3	6.5	3.78E−14	1.19E−06	17.9	1.93E−13	15.0	4.02E−14
R101	16.9	6.6	1.78E−14	2.28E−06	16.5	6.83E−12	14.9	4.44E−14
R102	16.4	6.0	1.52E−14	1.15E−06	17.1	1.62E−12	14.4	2.01E−14
R103	15.9	5.6	2.51E−15	2.54E−07	16.2	2.27E−14	14.4	1.59E−14
R104	17.0	6.4	1.59E−14	2.99E−06	16.9	1.00E−13	15.5	3.18E−14
R105	16.0	5.8	1.26E−14	1.64E−07	16.5	2.62E−13	15.0	3.42E−14
R106	16.7	5.8	2.86E−14	1.07E−07	17.0	1.61E−10	15.1	1.04E−13
R107	16.8	6.1	2.25E−14	7.95E−07	17.6	7.39E−14	14.7	6.20E−14
R108	17.2	5.9	4.49E−14	4.74E−06	17.0	5.33E−14	14.4	1.53E−13
R109	17.5	6.6	2.01E−14	5.59E−09	16.8	1.42E−14	14.0	3.34E−14
R200	58.8	20.0	1.14E−12	5.18E−05	68.6	2.08E−11	51.6	7.98E−12
R201	57.2	19.2	5.80E−12	1.12E−03	66.5	1.38E−11	53.6	3.72E−12
R202	57.3	19.8	5.85E−13	1.48E−04	64.0	2.14E−12	54.2	1.90E−12
R203	60.6	21.8	3.12E−13	2.83E−05	59.4	8.13E−13	52.8	1.39E−12
R204	60.4	21.8	1.28E−13	3.49E−04	62.0	2.76E−12	52.8	3.87E−13
R205	59.4	20.9	1.92E−13	1.90E−03	67.6	7.82E−13	52.7	8.66E−13
R206	59.1	21.1	1.58E−13	4.50E−05	64.8	8.26E−13	52.8	5.93E−13
R207	58.7	20.0	3.04E−13	3.44E−03	64.0	8.39E−13	53.9	3.10E−12
R208	59.0	21.2	5.69E−13	3.64E−05	63.1	2.27E−11	54.2	1.17E−12
R209	56.4	19.5	9.45E−13	7.85E−05	64.0	9.78E−12	52.8	1.94E−12
P9	18.7	7.7	2.88E−12	2.20E−01	13.3	3.40E−05	41.5	1.34E−02
P40	208.0	68.1	1.11E−16	5.31E−05	342.0	1.56E−11	1340.0	4.10E−14

From intermediate output not shown in the tables we determined the number of WDK iterations used by DROOTS after the initial approximations have been obtained by the modified Newton method. In more than half of all the problems two iterations were used. Otherwise the number ranged from 1 to 7, except for three notable exceptions. In Table II, 47 such iterations were used for polynomial 8, 28 for polynomial 9, and 72 for

Table II. Results Using DROOTS and Corresponding Programs from the IMSL and NAG Libraries on the Following Polynomials (Time is in Milliseconds, and Error is the Maximum Error Observed)

1: based on roots at 1.001, 0.998, 1.00002, 0.99999, and 0.1
2: based on roots at $0.1 \pm 10^{-5}i$, 10, 1, 0.1, 0.01, and 0
3: based on roots at $0.1 \pm 10^{-6}i$, 10, 1, 0.1, 0.01, and 0
4: $(z - 1)^2(z - 5i)^2(z + i)^3$
5: $(z - 1)^{10}$
6: based on $(z - 0.1)^4(z - 0.2)^3(z - 0.3)^2(z - 0.4)$
7: based on roots at $4 \pm 0.1i$, 10, 5, 4, 4, 3, 3, 2, 1
8: based on roots at 10^k, $k = -10, -9, \ldots, -1$
9: $(z - 3)^3(z + 1)^4(z + i)^2(z - 1 - 2i)(z - 1)$
10: based on roots at 8, 7, $6 \pm \alpha i$, $5 \pm \alpha i$, $4 \pm \alpha i$, $3 \pm \alpha i$, 2, 1, where $\alpha = 10^{-3}$
11: based on Wilkinson polynomial of degree 20 (roots at 1, . . ., 20)
12: roots at $-9.5 + a$, where $a = 0, 1, \ldots, 19$
13: $z^6(z + 10)^5(z - 10)^5(z + i)^2(z - i)^2$
14: based on roots at $re^{2\pi kj/30}$, $r = 1$ and $k = 0, \ldots, 14$, $r = 0.9$ and $k = 15, \ldots, 29$
15: 9 roots in 3 × 3 box at (10 . . . 12 by 1, 10 . . . 12 by 1)
16: 25 roots in 5 × 5 box at (−2 . . . 2 by 1, −2 . . . 2 by 1)
17: 49 roots in 7 × 7 box at (−3 . . . 3 by 1, −3 . . . 3 by 1)
18: $z^{48} - z^{47} - z^{46} - \cdots - z - 1$
19: $(z^{24} - z^{23} - z^{22} - \cdots - z - 1)^2$
20: $(z^{12} - z^{11} - z^{10} - \cdots - z - 1)^4$

	DROOTS				IMSL		NAG	
P(z)	Total Time	Time in Start	Final Error	Error in Start	Total Time	Final Error	Total Time	Final Error
1	6.9	3.9	6.04E−06	1.78E−03	9.3	1.03E−05	3.9	6.14E−05
2	11.4	4.8	2.12E−09	1.85E−04	7.8	9.13E−09	17.0	1.26E−08
3	12.0	4.8	5.91E−07	1.85E−04	8.2	4.47E−09	16.9	2.19E−06
4	19.0	6.9	1.71E−12	2.78E−03	18.1	2.46E−07	37.8	2.83E−05
5	7.4	1.9	4.10E−10	0.00E+00	8.9	8.69E−11	3.5	0.00E+00
6	24.1	10.1	5.01E−05	3.64E−04	17.5	2.96E−06	44.3	1.61E−04
7	15.7	6.0	4.70E−08	4.93E−02	21.2	4.65E−05	34.1	6.34E−04
8	123.0	21.2	1.39E−17	5.95E−04	10.7	1.39E−17	10.2	1.69E−21
9	126.0	12.4	1.31E−06	2.00E+00	24.2	7.23E−07	45.9	2.81E−04
10	21.8	8.2	1.53E−05	2.80E−02	38.3	1.99E−05	39.2	2.44E−05
11	44.2	15.7	4.91E−02	3.06E−02	62.7	7.10E−03	83.1	3.29E−02
12	44.9	15.0	1.46E−12	7.29E−01	71.2	2.81E−12	58.9	1.31E−12
13	1030.0	30.2	8.57E−04	2.60E−01	42.2	1.02E−02	84.7	1.35E−02
14	123.0	39.2	4.85E−12	2.40E−04	200.0	4.85E−12	163.0	4.85E−12
15	13.7	7.1	1.16E−05	8.18E−05	26.1	4.66E−02	34.6	9.00E−04
16	81.3	32.7	1.11E−16	5.44E−05	93.5	1.11E−15	64.2	1.01E−14
17	336.0	98.1	4.44E−16	1.55E−01	347.0	6.40E−14	218.0	2.32E−14
18	294.0	115.0	4.44E−16	3.36E−02	589.0	6.88E−07	376.0	1.31E−14
19	305.0	95.9	7.14E−10	2.38E−01	531.0	6.47E−07	577.0	7.34E−08
20	263.0	115.0	2.28E−10	4.27E−02	697.0	8.16E−03	591.0	5.81E−04

polynomial 13. DROOTS is much the slowest in these three cases, but it is much more accurate in the last case.

The results of this section show that a reasonably fast Newton start (with constraints) followed by WDK iterations (with special care taken especially near multiple roots or clusters of roots) makes an efficient combination for determining roots of polynomials to good accuracy.

10. CONCLUDING REMARKS

We have shown that DROOTS is a reasonably accurate and efficient program for finding all the roots of a polynomial, and we believe it may be particularly reliable. As indicated in the abstract it will be made available to anyone who wishes to have it, but it should be considered to be only a prototype. We plan to develop it further in a number of ways. For example, we plan to include special provision for low-degree polynomials ($n = 1, 2$, and possibly 3 or even 4). We have carried out some preliminary experiments in which the program uses the multiplicity test not just to test for convergence, but also to decide earlier if convergence is close enough that the approximations can be moved toward the mean before carrying out the next iteration. It might also help to avoid doing the convergence test at every step in the iteration process, and we hope to develop a more sophisticated way of determining the starting values.

We also plan to prepare a detailed report on the testing program, including a comprehensive list of test problems.

ACKNOWLEDGMENTS

Besides Catalina Trevizan and Margaret Solowiej, whom we have already mentioned, we would like to thank Edgar Smart and Tom Fairgrieve for their help in much of the earlier programming and testing. Tom Fairgrieve also developed most of our current test program, while Hisao Tamaki designed and implemented the algorithm for matching the true roots with the computed approximations to those roots. We also wish to thank the referees for some helpful suggestions, as well as Pierre Milman for stimulating discussions during the preparation of this article. We also very much appreciate the generosity of IMSL and NAG for giving us the Sun 4 versions of their root-finding programs.

REFERENCES

ABERTH, O. 1973. Iteration methods for finding all zeros of a polynomial simultaneously. *Math. Comput. 27*, 122, 339–344. See also Statement of Priority. *Math. Comput. 30*, 135 (Apr. 1976), 680.

ABERTH, O. 1988. *Precise Numerical Analysis.* William C. Brown, Dubuque, Iowa.

ADAMS, D. A. 1967. A stopping criterion for polynomial root finding. *Commun. ACM 10*, 10 (Oct.), 655–658.

DOCHEV, K. 1962. A modified Newton method for the simultaneous approximate calculation of all the roots of a given algebraic equation. *Phys. Math. J. Bulgarian Acad. Sci. 5*, 136–139. In Bulgarian.

DURAND, E. 1960. *Solutions Numériques des Équations Algébriques. Vol. 1, Équations du Type F(x) = 0, Racines d'un Polynôme.* Masson, Paris.

FREEMAN, T. L. 1989. Calculating polynomial zeros on a local memory parallel computer. *Parallel Comput. 12*, 351–358.

FREEMAN, T. L. AND BANE, M. K. 1991. Asynchronous polynomial zero-finding algorithms. *Parallel Comput. 17*, 673–681.

IMSL. 1985. *User's Manual.* International Mathematical and Statistical Libraries, Houston, Tex.

JENKINS, M. A. AND TRAUB, J. F. 1975. Principles for testing polynomial zerofinding programs. *ACM Trans. Math. Softw. 1*, 1 (Mar.), 26–34.

KERNER, I. O. 1966. Ein Gesamtschrittfahren zur Berechnung der Nullstellen von Polynomen. *Num. Math. 8*, 290–294.

MILOVANOVIĆ, G. V. AND PETKOVIĆ, M. S. 1986. On computational efficiency of the iterative methods for the simultaneous approximation of polynomial zeros. *ACM Trans. Math. Softw. 12*, 4 (Dec.), 295–306.

NAG. 1991. *Fortran Library Manual, Mark 15A.* Numerical Algorithms Group Ltd., Oxford.

PASQUINI, L. AND TRIGIANTE, D. 1985. A globally convergent method for simultaneously finding polynomial roots. *Math. Comput. 44*, 169 (Jan.), 135–149.

PETERS, G. AND WILKINSON, J. H. 1971. Practical problems arising in the solution of polynomial equations. *J. Inst. Math. Appl. 8*, 16–35.

SMITH, B. T. 1970. Error bounds for zeros of a polynomial based upon Gerschgorin's theorems. *J. ACM 17*, 4 (Oct.), 661–674.

SOLOWIEJ, M. 1983. N-dimensional Newton method with precision control for determining roots of complex polynomials. M.Sc. thesis, Dept. of Computer Science, Univ. of Toronto, Toronto, Canada.

TREVIZAN, C. A. 1977. A zero finding algorithm for complex polynomials. M.Sc. thesis, Dept. of Computer Science, Univ. of Toronto, Toronto, Canada.

WEIERSTRASS, K. 1891. Neuer Beweis des Satzes, dass jede ganze rationale Function einer Veränderlichen dargestellt werden kann als ein Product aus linearen Functionen derselben Veränderlichen. Reproduced in *Ges. Werke 3* (1903), 251–269.

WILKINSON, J. H. 1963. *Rounding Errors in Algebraic Processes.* Prentice-Hall, Englewood Cliffs, N.J.

Received October 1993; revised January 1995, March 1995, and July 1995; accepted August 1995