PRINCIPLES FOR TESTING POLYNOMIAL ZEROFINDING PROGRAMS

M. A. Jenkins
Queen's University

J. F. Traub
Carnegie-Mellon University

Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pa.

March 1974

To be presented at Mathematical Software II at Purdue University (May, 1974)

J. F. Traub was supported in part by the National Science Foundation under Grant GJ32111 and the Office of Naval Research under Contract N0014-67-A-0314-0010, NR 044-422. M. A. Jenkins was supported in part by the Defense Research Board of Canada under Grant 9540-32, and by the National Research Council of Canada under Grant A7892.

ABSTRACT

The state-of-the-art in polynomial zerofinding algorithms and programs is briefly summarized. Our focus in this paper is on principles for testing such programs. We view testing as requiring four steps: (1) testing program robustness, (2) testing for convergence difficulties, (3) testing for specific weakness of the algorithms, (4) assessment of program performance by statistical testing.

We emphasize that the statistical testing must be done with care. There are many ways to generate "random" polynomials. Two classes of random polynomials which have been widely utilized are of only limited usefulness in terms of evaluating reliability or performance because they produce polynomials with very similar characteristics. We discuss classes of random polynomials which should be used.

The authors' algorithm for real polynomials has been reprogrammed with substantial modifications to the decision making processes. In order to verify that the new program is a reliable and efficient mathematical software product, it underwent the testing outlined above. The results of part of this testing are described in this paper.

1. Introduction

The numerical solution of polynomial equations has been the subject of intensive research in recent years with proposals of new algorithms appearing frequently in the numerical analysis literature. While many of the algorithms which are proposed are mathematically interesting only a few are serious contenders as the basis of fast, reliable, mathematical software for this problem. The purpose of this paper is to propose means of testing programs for polynomial zerofinding in order to separate those which should be recommended as contenders for inclusion in program libraries and those which should be considered only mathematical novelties. The emphasis here is on the testing of a single program which implements a particular algorithm, but similar techniques may be used for comparing the relative merits of several algorithms.

The polynomial zerofinding problem is a difficult computational problem. It can be viewed as a non-linear problem, or in terms of linear algebra as a special case of a non-Hermitian eigenvalue problem with (in general) non-linear divisors. Many algorithms which work easily on examples with well-separated simple zeros, fail on more difficult examples.

We assume here that we are discussing a program which is intended to find all the zeros of an arbitrary polynomial with real or complex coefficients. The current state-of-the-art is such that all the zeros can be found with $0(n^2)$ arithmetic operations using globally convergent techniques [Jenkins 1969, Smith 1967b]. Hence, to be competitive, a program must be based on an algorithm which is $0(n^2)$.

Algorithms can be classified as those which find a single zero or a pair of zeros at a time or those which determine all the zeros simultaneously. The former require the use of some deflation technique, either explicit or implicit

[Wilkinson 1963], to remove the zeros already determined. Most of the algorithms proposed for this problem use one of the following basic approaches or some combination of them:

(i) function iteration. An iteration function [Ostrowski 1966, Traub 1964] generates a sequence of approximations which converge to a zero. Common choices are Newton's method, Bairstow's method, Muller's method (all described in Householder [1970]), Laguerre's method [Smith 1967a], and many others. There are also versions of some of these iterations allowing all the zeros to be found simultaneously [Grau 1971].

(ii) separation of the zeros by powering. An algorithmic process produces a sequence whose elements are related to high powers of the zeros. One or more of the zeros are then estimated. Well known algorithms of this type are Bernoulli's method, and variations [Jenkins 1973], Graffe's method [Bareiss 1967] and the QD algorithm [Henrici and Watkins 1965].

(iii) topological methods. The complex plane is divided into regions and tests are made to determine which regions contain or exclude zeros. The regions containing zeros are themselves subdivided repeatedly until a zero is determined "accurately". Some of the techniques isolate a single zero [Lehmer 1961], while others find all the zeros simultaneously [Henrici and Gargantini 1967].

(iv) minimization techniques - zeros are found by finding the minimum of $|F|^2$ or $|F|$ using minimization algorithms which take special advantage

of the properties of polynomials [Ostrowski 1967], [Nickel 1966].

(v) factorization techniques - The polynomial is factored into two or more factors by some criteria. One approach is to use Euclid's algorithm to break the polynomial into factors each of which has simple zeros [Dunaway 1974]. Another factorization technique [Stewart 1969] requires all factors corresponding to a multiple zero to be in one of the factors.

The proceedings of the Ruschlikon symposium [Dejon and Henrici 1969] provide an illustration of the diverse approaches which have been applied to this fundamental problem.

Our experience in testing polynomial zerofinding programs has been gained while developing algorithms which use a combination of approaches (ii) and (i) [Jenkins and Traub 1970a, 1970b] to find linear or quadratic factors, and use explicit deflation to reduce to a smaller problem. To some extent this view of the zerofinding process biases our presentation of the principles for testing; however we attempt to discuss the issues in general terms.

2. Objectives

We are concerned with foolproof algorithms and programs for the following mathematical problem. Given the degree n, and the coefficients $a_0, a_1, \ldots, a_n$ of the polynomial

$$P(z) = a_0 z^n + a_1 z^{n-1} + \ldots + a_n, \text{ with } a_0 \neq 0,$$

we are to find the number of distinct zeros, k, the zeros $\alpha_1, \alpha_2, \ldots, \alpha_k$, and the corresponding multiplicities $m_1, m_2, \ldots, m_k$ such that P(z) is expressed

in factored form as

$$P(z) = a_0(z-\alpha_1)^{m_1}\ldots(z-\alpha_k)^{m_k} \quad .$$

In computational work, the sharp distinction between multiple zeros and closely clustered distinct zeros is blurred and we usually settle for finding n approximations $\tilde{\alpha}_1, \tilde{\alpha}_2, \ldots, \tilde{\alpha}_n$ to the zeros which satisfy at least one of the following two objectives:

(i) a set of approximations $\tilde{\alpha}_i$ which are "accurate" approximations of the true zeros of the polynomial.

(ii) a set of approximations $\tilde{\alpha}_i$ such that the polynomial

$$\tilde{P}(z) = a_0(z-\tilde{\alpha}_1)\ldots(z-\tilde{\alpha}_n)$$

formed by multiplying the linear factors together in finite-precision floating-point arithmetic produces "accurate" approximations of the coefficients of the polynomial.

We say, somewhat arbitrarily, that a polynomial zerofinding program is <u>successful on a particular example</u> if the approximations it produces satisfy <u>either</u> of the objectives (i) or (ii).

The accuracy which one may expect to achieve in calculating zeros is limited by the condition of these zeros [Wilkinson 1963]. In particular, for multiple zeros, perturbations of size ε in the coefficients cause perturbations of size $\varepsilon^{\frac{1}{m}}$ in the zeros. If the objective of the program is to obtain "accurate" approximations, it may be desirable to measure the degree of success by computing error bounds for the computed zeros (see Section 7). The second objective is more easily realized and can be readily tested against the input data by

forming $\tilde{P}$. (A meaningful way to compare $\tilde{P}$ with P is described in Section 7.)

3. The Testing Process

The tests we propose for a zerofinding program do not guarantee that the program will always be successful in the above sense; nor is it implied that these are the only valid tests for polynomial zerofinding programs. However, they do provide for a systematic rational means of evaluating programs in order to determine if they are potentially useful as mathematical software for the zerofinding problem.

We view the testing of a program as requiring four distinct steps.

(i) testing program robustness - done by examination of the program or by testing with pathological examples,

(ii) testing for convergence difficulties - done by a combination of examination of the program and by applying the program to specific test examples,

(iii) testing for specific known weaknesses of the algorithm(s) used - done by applying the program to test examples which are selected to illustrate specific potential flaws in the algorithm,

(iv) assessment of the performance of the program in terms of reliability and efficiency - done by gathering statistics on the program using randomly generated polynomials.

We discuss procedures for carrying out these steps in the succeeding sections.

4. Checking Program Robustness

By program robustness we mean the ability of a program to degrade gracefully near the boundary of the problem space where the algorithm applies. A more specific definition is given in [Cody 1971] where the problem is discussed in a more general framework. Appropriate questions to ask for the zerofinding problem include whether the program:

(i) tests if a leading coefficient is zero or "nearly" zero. If it is exactly zero the degree of the polynomial is incorrect. A warning that this condition has arisen should be issued and then either the reduced problem can be solved or the computation terminated. A very small leading coefficient combined with large coefficients for some of the other terms may imply that the polynomial has a zero or zeros outside the upper limit of the exponent range. A result of "machine infinity" may be considered an acceptable result for some purposes.

(ii) tests if a trailing coefficient is zero, or "nearly" zero. If it is exactly zero, there is a zero of the polynomial at the origin. It can be detected directly and the problem reduced to one of lower degree. It is generally safer (and certainly faster) to do this directly than to count on the zerofinding algorithm eventually discovering it. A very small trailing coefficient combined with large coefficients for some of the other terms may imply that the polynomial has a zero or zeros outside the lower limit of the exponent range.

(iii) handles low degree cases properly. If a quadratic solver is used it should be programmed according to the principles espoused by [Kahan

1966] and [Forsythe 1967]. A test should be made for the degenerate cases with degree less than or equal to 1.

(iv) scales coefficients to avoid exponent underflow and overflow difficulties. The polynomial, the algorithm, or both may have to be scaled to reduce the likelihood of overflow or underflow interfering with the calculation. Scaling techniques are discussed in [Smith 1967a], [Jenkins 1969] and [Dunaway 1974]. If scaling is done it is desirable that a power of the base of the floating-point system for the computer be used.

(v) specifies a finite computation. There should be reasonable limits on all iterations or loops. The limits are essential for functional iteration algorithms since many of them can cycle indefinitely on some examples [Smith 1967a, Ray 1966, Brolin 1965].

(vi) provides a failure exit or a testable failure flag. These should be included in the calling sequence to allow for unusual input or failure of the algorithm. The very appearance of a failure parameter makes the user think about what to do if the calculation fails to succeed.

5. Testing for Convergence Difficulties

Generally, polynomial zerofinding algorithms determine a sequence (or a set of sequences) which hopefully converge to a zero (or a set of zeros) of the given polynomial. There are two basic computational problems which arise: first, if the sequence is converging satisfactorily how do we decide when to terminate the sequence and accept the value obtained; second, if the sequence is not converging, or is converging very slowly, how do we detect this and what

do we do. (The problem is somewhat different for topological algorithms; we shall not pursue this here.)

We consider first the problem of developing a criterion for terminating a sequence of iterates $\{x_i\}$ which are converging to a zero of the given polynomial. The more obvious criteria have weaknesses. If ϵ is fixed, the test

$$\frac{|x_{i+1}-x_i|}{|x_{i+1}|} < \epsilon$$

may fail to terminate if the zero is ill-conditioned. If ϵ is fixed, a residual test such as

$$|P(x_i)| < \epsilon$$

may fail to terminate or may terminate too early if the polynomial is poorly scaled.

One satisfactory means of terminating a converging sequence of iterates is to stop when the polynomial value becomes dominated by the roundoff error in evaluating the polynomial. This is a residual test in which the appropriate ϵ is chosen by monitoring the evaluation roundoff error [Peters and Wilkinson 1971, Adams 1967, Kahan 1966]. Even this technique must be used with care for if underflow occurs in the evaluation and the result is replaced by zero, the roundoff error analysis is invalid and an inaccurate approximation to a zero may be accepted. The underflow difficulty may be avoided by scaling [Smith 1967a, Jenkins 1969]. Other termination criteria involving the sequence of residuals or the sequence of differences of the iterates may also be used.

The type of termination criteria used in the program can be checked by reading the program or by testing with some specific polynomials. For example the cubic polynomial

$$P_1(z) = B(z^3-z^2-A^2z+A^2) = B(z-A)(z+A)(z-1)$$

can be used with A large and small to test that large and small zeros do not cause the termination criterion to fail and with B large and small to ensure that large and small polynomial coefficients do not similarly cause failure. To test for a suitable convergence criterion for an ill-conditioned polynomial, we suggest the polynomial

$$P_2(z) = \prod_{i=1}^{r} (z-i)$$

with zeros at 1, 2, ..., r where r is chosen small enough that the coefficients of the polynomial are exactly representable in the precision used [Wilkinson 1963]. Underflow in polynomial evaluation is likely in a polynomial

$$P_3(z) = \prod_{i=1}^{r} (z-10^{-i})$$

with zeros at 10^{-1}, 10^{-2}, ..., 10^{-r} where r is chosen small enough that the constant term does not underflow. Such a polynomial may be used to test whether a termination criterion based on roundoff error analysis fails. For other termination criteria, it is a challenge to the creativity of the program tester to find examples which thoroughly test the particular criterion.

The second problem, that of detecting and curing slow convergence or non-convergence of the sequence of iterates, is difficult to discuss in general as the difficulties are often algorithm-dependent. The obvious cure is to limit the length of the sequence a priori; however, what appropriate remedial action should be taken depends greatly on the algorithm used.

Multiple zeros or nearly multiple zeros cause convergence difficulties for many algorithms. As noted in Section 2, such multiple zeros are inherently ill-conditioned and one can expect to calculate a zero of multiplicity m only to a relative accuracy of roughly $\epsilon^{\frac{1}{m}}$ where ϵ is the relative precision of the computer arithmetic [Wilkinson 1963]. However, in some recent work Dunaway [Dunaway 1974] claims that multiple zeros can be calculated with higher accuracy. In addition for many algorithms, multiple zeros result in a considerably slower

rate of convergence and as a result, the time required for such examples may be quite long compared to other examples of the same degree. Some test polynomials for checking the performance of a program on multiple and nearly multiple zeros are

$$P_4 = (z-.1)^3(z-.5)(z-.6)(z-.7),$$

$$P_5 = (z-.1)^4)z-.2)^3(z-.3)^2(z-.4),$$

$$P_6 = (z-.1)(z-1.001)(z-.998)(z-1.00002)(z-.99999),$$

$$P_7 = (z-.001)(z-.01)(z-.1)(z-.1+Ai)(z-.1-Ai)(z-1)(z-10),$$

with A chosen to be 0, 10^{-10}, 10^{-9}, 10^{-8}, 10^{-7}, 10^{-6},

$$P_8 = (z+1)^5 \quad .$$

Equimodular zeros cause convergence difficulties for algorithms which use powering techniques to separate zeros, either for obtaining a starting value or as the main algorithm. The case of equimodular zeros must be handled in some special way and if not done carefully, it may result in an inefficient algorithm or one which fails. Test examples are P_8 above and

$$P_9 = (z^{10}-10^{-20})(z^{10}+10^{20})$$

which has two sets of equimodular zeros.

6. Testing for Defects in a Program Implementing a Particular Algorithm

The testing for defects in a program implementing a particular algorithm is a creative challenge to those engaged in the art of program testing. We mention some general approaches which might be taken. First we suggest that the mathematical algorithm be examined for possible occurrences of "undefined"

results. If these are mathematically possible, the defense mechanisms in the program to avoid such occurrences should be examined. Often an obvious way to "beat" the defense can be found and an example constructed which produces the appropriate conditions to cause the "undefined" result to appear. A second approach is to study the decision mechanisms of the program. If there is a key decision (such as switching between a slowly convergent starting process and a rapidly converging iteration) which if made incorrectly causes the algorithm to fail, then an example which forces a wrong decision can be sought. A third approach is to examine the roundoff error properties of the basic algorithm to determine if the algorithm is numerically stable. Such an analysis may suggest an example in which the roundoff error destroys the accuracy of the computed zeros. [Wilkinson 1963] provides a case study involving Graeffe's method.

For most algorithms which find the zeros one or two at a time, either explicit or implicit deflation must be used. For either choice, the algorithm is of interest only if we can ensure that the remaining zeros are found to the accuracy one would expect. The advantage of explicit deflation is that the resulting zeros almost always satisfy objective (ii) of Section 2 if the deflation has been achieved in a stable manner. Wilkinson showed [Wilkinson 1963] that deflation using the Horner recurrence is unstable if a zero considerably larger in modulus than the smaller zeros is removed first. As a result many of the algorithms which use deflation control the order in which the zeros are found [Smith 1967a, Jenkins 1969].

A composite deflation technique [Bingham 1967, Peter and Wilkinson 1971] allows any size zero to be removed without severely affecting the accuracy of the remaining zeros. Explicit deflation is at best a semi-stable process in the sense that the cumulative effect after many steps may result in a significant

loss of accuracy. This is particularly true if one of the reduced polynomials is considerably more ill-conditioned than the original polynomial. In order to test for deflation stability we suggest the examples

$$P_{10}(z) = (z-A)(z-1)(z-A^{-1}),$$

with $A = 10^3$, 10^6, or 10^9, and

$$P_{11}(z) = \prod_{k=1-M}^{M} (z-e^{\frac{ik\pi}{2M}}) \prod_{k=M}^{3M} (z-.9e^{\frac{ik\pi}{2M}})$$

with $M = 15, 20, 25$.

In the first example if the zero near 1 or the zero near A is found first, the zero at A^{-1} will be very inaccurate if ordinary deflation is used, although the composite deflation process will produce accurate zeros. In the second example the polynomial has zeros which lie on a semicircle of radius 1 in the right half of the complex plane and one of radius .9 in the left half-plane. Each of the polynomials corresponding to the zeros on one of the half-circles is quite ill-conditioned (in the sense that some of the zeros are ill-conditioned); however the same zeros are well-conditioned in the original polynomial. Thus an algorithm which tends to find the zeros of modulus .9 first would do poorly on this example. This specific example [Businger 1969] was constructed to show that the Jenkins-Traub algorithms can fail to succeed on an "innocent" looking example. Since there is no a priori technique for detecting the presence of an ill-conditioned factor, any algorithm using explicit deflation is susceptible to this difficulty.

Implicit deflation [Wilkinson 1963] is also unsatisfactory in some respects. After the first step one must work with a rational function instead of a polynomial

and hence techniques which depend on using the polynomial form are ruled out. Computationally the main difficulty occurs with multiple or closely clustered zeros where both the numerator and denominator of the rational function are approaching zero. Tests such as those we suggested in Section 5 for multiple zeros can be used to test programs which use implicit deflation.

7. Assessment of Performance

A library program for polynomial zerofinding should ideally find all the zeros of an arbitrary polynomial within the objectives of Section 2, at a predictable cost. By doing tests with randomly generated polynomials we can gather statistics on the performance of the program with respect to reliability, accuracy and cost prediction. (We could also test with specific polynomials but shall confine ourselves to testing with random polynomials here.) In order to do this adequately we need:

(i) a meaningful measure of cost,

(ii) a measure of accuracy.

Cost can be measured in terms of execution time, polynomial evaluations or equivalents, or arithmetic operations. If these measures are to be used in comparing algorithms, then great care must be taken in gathering the statistics and analyzing the results. Timing measurements are valid for comparisons only if accurate timings are available and if the algorithms are coded with the same level of optimization. However, if valid, they do yield realistic cost estimates. Counts of polynomial evaluations or arithmetic operations may not reflect the actual cost of executing the algorithm due to the amount of overhead involved in loop control, array access, etc. and are useful in comparisons only for detecting gross differences between programs.

The accuracy of the results should be tested relative to the objective of the program. If the objective is simply to find as accurate zeros as can be expected, the relative error in the zero should be estimated. If the zeros are not known, the accuracy of the computed zeros will have to be estimated by a posteriori bounding techniques [Champagne 1964, Jenkins 1969, Smith 1970]. If the objective is to reproduce the polynomial the accuracy of the reconstructed polynomial has to be tested carefully. Testing the quantities

$$\frac{a_j - \tilde{a}_j}{\underline{a}_j} \quad , \quad j=0,\ldots,n$$

where the a_j's are the original coefficients, the $\tilde{a}_j$'s are the reconstructed coefficients, and the $\underline{a}_j$'s are the coefficients constructed using the quantities $|\text{Re } \tilde{\alpha}_j|+1|\text{Im } \tilde{\alpha}_j|$ allows approximations to zero coefficients in the original polynomial to be tested in a meaningful way [Kahan 1966].

We have recommended above that the reliability, accuracy, and efficiency of zerofinding program be estimated by gathering statistics on the performance of the program on randomly generated polynomials. There are many ways to generate random polynomials and particular choices may affect the quality of the performance evaluation greatly. Several classes of random polynomials are listed below. The first two classes have been widely used but are of limited usefulness in terms of evaluating reliability or performance because they produce polynomials with very similar characteristics. The remaining classes produce polynomials with less similarity from one example to another.

(1) polynomials constructed from random zeros from a uniform distribution. This is a poor choice as the randomness "averages out" in the coefficients and the polynomials differ but little from each other.

(ii) polynomials whose coefficients are chosen randomly from a uniform distribution. This is also a poor choice, since the zeros tend to be uniformly distributed around the origin near the unit circle.

(iii) polynomials constructed from zeros with some zeros chosen from a wide uniform distribution and the remainder chosen from a much narrower distribution. By decreasing the width of the second distribution and by increasing the percentage of the zeros chosen from this distribution, polynomials of increasing ill-condition can be generated. This is a reasonable test for the behavior of the program on clustered zeros.

(iv) polynomials whose coefficients are chosen randomly by taking the mantissa and exponents from separate uniform distributions. The resulting polynomials have widely varying zeros and hence yield a reasonable test that the program has wide applicability.

(v) polynomials constructed from zeros chosen in the manner of the coefficients of (iv). This results in zeros varying greatly in size and the corresponding polynomials often illustrate inefficiencies in the program which may not be apparent from other tests. For example poorly chosen initial iterates may tend to "wander" slowly to the area of the zero and such behavior will be magnified on this class of polynomials.

With these techniques one can generate real or complex polynomials as appropriate for the program being tested. We recommend that only a few tests with (i) and (ii) be done for comparison with the other results in the literature and that the major effort be concentrated on (iii), (iv) and (v).

8. Examples of Testing

Several examples of testing zerofinding programs have been reported in the literature including tests using randomly generated polynomials. [Dejon and Nickel 1967, Henrici and Watkins 1965, Jenkins 1969]. Tests on specific examples accompany almost every paper in this field; however, the tests used on new algorithms are all too often chosen to indicate that the particular algorithm works well on a particular class of problems. There have been some reports of comparative evaluations [Witte 1967, Smith 1967a, Dodson et al. 1968] which have used specific examples to test weaknesses of available programs. The testing by Smith [1967b] and Jenkins [1969] does indicate that it is possible to develop fast, accurate, reliable zerofinding programs if one is willing to pay the cost of having a complicated program.

The testing procedure described in this paper was developed in conjunction with the implementation of the authors' variable-shift algorithms [Jenkins and Traub 1970a, 1970b]. The program for complex polynomials has been published [Jenkins and Traub 1972]. The program for real polynomials has been recently modified in order to improve the decision making processes [Jenkins 1974]. In order to verify that the new program is a reliable and efficient mathematical software product the testing procedure described in this paper was applied.

The results of the tests with specific examples are summarized in Table I. The tests were carried out using double precision calculations on a Burroughs B6700 with 26 octal ($\simeq$ 23 decimal) digit arithmetic. The algorithm performed well with respect to objective (ii), in that in all cases the factorization could reproduce the coefficients to about single precision accuracy and in many cases to nearly double-precision accuracy. This is expected for explicit deflation [Wilkinson 1963]. However, in respect to objective (i), the zerofinder

Table I - Tests on Specific Polynomials

		Example		n	δ_1	δ_2	Time (Milliseconds)	Time/n^2
Termination	P_1	$A=10^{20}$	$B=1$	3	2.8×10^{-23}	1.0×10^{-23}	20	2.2
		$A=10^{-20}$	$B=1$	3	3.0×10^{-23}	1.0×10^{-23}	36	4.0
		$A=.1$	$B=10^{40}$	3	4.5×10^{-23}	1.0×10^{-23}	44	4.9
		$A=.1$	$B=10^{-40}$	3	1.8×10^{-23}	1.0×10^{-23}	33	3.7
	P_2	$R=20$		20	1.1×10^{-19}	2.4×10^{-10}	952	2.4
	P_3	$R=9$		9	3.6×10^{-23}	2.1×10^{-23}	267	3.3
Behaviour on Multiple or Clustered Zeros	P_4			6	2.6×10^{-18}	1.7×10^{-10}	104	2.9
	P_5			10	1.5×10^{-18}	4.7×10^{-6}	269	2.7
	P_6			5	6.9×10^{-11}	1.3×10^{-4}	79	3.2
	P_7	$A=0$		7	5.5×10^{-15}	3.8×10^{-9}	127	2.6
		$A=10^{-10}$		7	2.0×10^{-14}	3.8×10^{-9}	130	2.7
		$A=10^{-9}$		7	1.0×10^{-15}	3.8×10^{-9}	124	2.5
		$A=10^{-8}$		7	9.4×10^{-19}	3.8×10^{-9}	123	2.5
		$A=10^{-7}$		7	1.2×10^{-21}	3.5×10^{-11}	128	2.6
		$A=10^{-6}$		7	3.8×10^{-23}	1.1×10^{-12}	127	2.6
Behaviour on Equimodular Zeros	P_8			25	1.4×10^{-20}	2.5×10^{-5}	1859	3.0
	P_9			20	2.7×10^{-22}	1.0×10^{-22}	1671	4.2
Deflation Stability	P_{10}	$A=10^{3}$		3	1.1×10^{-22}	2.6×10^{-23}	24	2.7
		$A=10^{6}$		3	2.6×10^{-23}	1.3×10^{-23}	25	2.8
		$A=10^{9}$		3	4.3×10^{-23}	0	24	2.7
	P_{11}	$M=15$		60	5.2×10^{-19}	1.0×10^{-10}	9752	2.7
		$M=20$		80	6.3×10^{-12}	2.0×10^{-3}	24688	3.9
		$M=25$		100	5.5×10^{-14}	8.0×10^{-2}	42581	4.3

δ_1=largest relative error in the reconstructed coefficients
δ_2=largest relative error in the computed zeros.

had difficulty with P_{11}, the polynomial with an ill-conditioned sub-polynomial. The accuracy of the zeros in the multiple and clustered zeros cases is about as good as we would expect for this class of method. We feel that the uniformity in the timing estimates, expressed as a function of n^2, is quite striking.

We now turn to a discussion of the testing of the algorithm for real polynomials using random polynomials which were generated for each of the five classes described in Section 7. The specific tests were:

(i) (a) zeros chosen in the region $\{Z: |Re(Z)|<1, |Im(Z)|<1\}$

(b) zeros chosen in the interval (-1,1)

(ii) coefficients chosen in the interval (-1,1)

(iii) (a) n-k zeros chosen in the region $\{Z: |Re(Z)|<1, |Im(Z)|<1\}$ and k zeros chosen in the region $\{Z: |1-Re(Z)|<.1, |Im(Z)|<.1\}$

(b) n-k zeros chosen in the interval (-1,1), k zeros chosen in (.9,1.1)

(iv) coefficients of the form $x*10^e$ with x in (-1,1) and e in (-R,R), with R=5,10,15 and 20

(v) (a) zeros of the form $x_1 \times 10^{e_1} \pm i x_2 \times 10^{e_2}$ with x_1, x_2 in (-1,1) and e_1, e_2 in (-R,R) with R=1,2,3,4,5

(b) zeros of the form $x \times 10^e$ with x in (-1,1) and e in (-R,R) with R=1,2,3,4,5

The results of the tests are summarized in Table II for polynomials of degree 10. In this testing we measured the relative error in the reproduced coefficients for all cases and the relative error in the zeros for those polynomials

Table II - Random Polynomials of Degree 10 with 50 Cases Each

Test		Failures	δ_1	δ_2	Max time (Milliseconds)	Average Time (Milliseconds)	Average time / n^2
(i)	(a)	0	4.2×10^{-21}	1.2×10^{-16}	433	335	3.4
(i)	(b)	0	3.2×10^{-20}	5.3×10^{-18}	1286	399	4.0
(ii)		0	1.1×10^{-20}	-	591	324	3.2
(iii)	(a) K=4	0	5.6×10^{-21}	8.9×10^{-17}	589	332	3.3
	K=6	0	1.0×10^{-16}	3.9×10^{-13}	454	330	3.3
	K=8	0	7.2×10^{-15}	1.6×10^{-11}	395	314	3.1
(iii)	(b) K=4	0	3.8×10^{-18}	1.9×10^{-18}	3502	657	6.6
	K=6	0	8.4×10^{-19}	6.3×10^{-16}	2710	532	5.3
	K=8	0	3.4×10^{-20}	2.0×10^{-15}	2111	470	4.7
(iv)	R=5	0	1.1×10^{-20}	-	742	301	3.0
	R=10	0	6.0×10^{-21}	-	4495	464	4.6
	R=15	1	1.8×10^{-20}	-	4850	420	4.2
	R=20	1	3.9×10^{-21}	-	12990	738	7.4
(v)	(a) R=1	0	4.2×10^{-21}	1.2×10^{-16}	326	273	2.7
	R=2	0	4.0×10^{-21}	9.1×10^{-18}	468	326	3.3
	R=3	0	1.3×10^{-14}	9.4×10^{-14}	739	325	3.3
	R=4	0	1.1×10^{-21}	7.1×10^{-20}	423	282	2.8
	R=5	0	2.2×10^{-21}	1.7×10^{-19}	414	272	2.7
(v)	(b) R=1	0	1.2×10^{-20}	2.7×10^{-19}	1402	390	3.9
	R=2	0	1.9×10^{-18}	2.6×10^{-17}	1964	361	3.6
	R=3	0	1.9×10^{-17}	4.7×10^{-15}	1843	363	3.6
	R=4	0	1.7×10^{-17}	5.2×10^{-12}	581	230	2.3
	R=5	0	1.6×10^{-16}	9.5×10^{-11}	1111	266	2.7

generated from random zeros. Again we note the uniformity of the timing as a function of n^2.

The zerofinding program failed for two examples of the 200 random polynomials of Type (iv) tested. As the timing data indicates the program had more difficulty with examples of this type and with examples of Type (iii)(b) than the other classes of polynomials. For all polynomials solved, the reconstructed coefficients were accurate to at least single precision and in most cases to much more. Similar results were observed for polynomials of higher degree.

9. Conclusions

The four stages of testing a program for polynomial zerofinding should be considered sequential tasks with an elaborate performance evaluation carried out only if the program appears sound after the specific tests have been done. If the program does poorly on the tests of Sections 5 and 6 it may not be that the basic algorithm is poor, but that the program author is unaware of some of the common problems which can arise. We recommend the excellent paper by [Peters and Wilkinson 1971] as a good source of practical tips on avoiding these problems.

We feel that the polynomial zerofinding area has reached a level of maturity where good mathematical software does exist and criteria for discriminating between good and poor programs are reasonably well known. It seems to us a waste of effort for someone to publish a new polynomial zerofinding program unless it is at least competitive with the best of those available and/or it has important and relevant features that published programs do not have. A requirement for publication of a new algorithm should be a thorough evaluation of the reliability and efficiency of the program carried out by someone other than its author.

References

[Adams 1967] D. A. Adams, A stopping criterion for polynomial root finding, Comm. ACM 10, pp. 655-658.

[Bareiss 1967] E. H. Bareiss, The numerical solution of polynomial equations and the resultant procedures, in Mathematical Methods for Digital Computers, Volume II, Ralston and Wilf (editors). John Wiley and Son, pp. 185-214.

[Bingham 1967] J.A.C. Bingham, An Improvement to Iterative Methods of Polynomial Factorization, Comm. ACM 10, pp. 57-60.

[Brolin 1965] Hans Brolin, Invariant sets under iteration of rational functions, Arkiv för Mathematik 6, pp. 103-144.

[Businger 1969] P. A. Businger - private communication.

[Champagne 1964] W. P. Champagne, On finding roots of polynomials by hook or by crook. Interim Technical Report No. 3, University of Texas Computation Center, Austin, Texas.

[Cody 1971] W. J. Cody, Software for the elementary functions, in Mathematical Software, J. R. Rice (editor). Academic Press, pp. 171-186.

[Dejon and Henrici 1969] B. Dejon and P. Henrici, editors, Constructive Aspects of the Fundamental Theorem of Algebra, Wiley-Interscience.

[Dejon and Nickel 1967] B. Dejon and K. Nickel, A never failing fast convergent root-finding algorithm, in [Dejon and Henrici 1969].

[Dodson et al. 1968] D. S. Dodson, P. A. Miller, W. C. Nylen, J. R. Rice, An evaluation of five polynomial zerofinders, Report CSD TR 24, Computer Science Department, Purdue University.

[Dunaway 1974] D. K. Dunaway, Calculation of zeros of a real polynomial through factorization using Euclid's algorithm. To appear SIAM J. Num. Anal.

[Forsythe 1967] G. E. Forsythe, What is a satisfactory quadratic equation solver?, in [Dejon and Henrici 1969].

[Grau 1971] A. A. Grau, The simultaneous Newton improvement of a complete set of approximate factors of a polynomial. SIAM J. Numer. Anal. 8, pp. 425-437.

[Henrici and Watkins 1965] P. Henrici and B. O. Watkins, Finding zeros of a polynomial by the Q-D algorithm, Comm. ACM 8, pp. 570-574.

[Henrici and Gargantini 1967] P. Henrici and I. Gargantini, Uniformly convergent algorithms for the simultaneous determination of all zeros of a polynomial, in [Dejon and Henrici 1969].

[Householder 1970] A. S. Householder, The Numerical Treatment of a Single Nonlinear Equation, John Wiley and Sons.

[Jenkins 1969] M. A. Jenkins, Three-stage variable-shift iterations for the solution of polynomial equations with a posteriori error bounds for the zeros. Diss., Stanford U., Stanford, Calif.

[Jenkins 1973] M. A. Jenkins, Bernoulli's method with implicit shifting, J. ACM 20, pp. 539-544.

[Jenkins 1974] M. A. Jenkins, Zeros of a real polynomial, an algorithm submitted to Comm. ACM.

[Jenkins and Traub 1970a] 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, pp. 252-263.

[Jenkins and Traub 1970b] M. A. Jenkins and J. F. Traub, A three-stage algorithm for real polynomials using quadratic iteration, SIAM J. Numer. Anal. 7, pp. 545-566.

[Jenkins and Traub 1972] M. A. Jenkins and J. F. Traub, Algorithm 419 - Zeros of a complex polynomial, Comm. ACM 15, pp. 97-99.

[Kahan 1966] W. Kahan, Numerical analysis lectures, Stanford, California.

[Lehmer 1961] D. H. Lehmer, A machine method for solving polynomial equations, J. ACM 8, pp. 151-162.

[Nickel 1966] K. Nickel, Die numerische Berechnung der Wurzeln eines Polynoms, Numer. Math 9, pp. 80-98.

[Ostrowski 1966] A. M. Ostrowski, Solutions of Equations and Systems of Equations (revised), Academic Press.

[Ostrowski 1967] A. M. Ostrowski, A method for automatic solution of algebraic equations, in [Dejon and Henrici 1969].

[Peters and Wilkinson 1971] G. Peters and J. H. Wilkinson, Practical problems arising in the solution of polynomial equations, J. Inst. Maths. Applics. 8, pp. 16-35.

[Ray 1966] T. L Ray, Laguerre's method for finding complex roots, Ph.D. Thesis, Stevens Institute of Technology, Hoboken, New Jersey.

[Smith 1967a] B. T. Smith, A zerofinding algorithm using Laguerre's method, M.Sc. thesis, U. of Toronto.

[Smith 1967b] B. T. Smith, ZERPOL, a zerofinding algorithm for polynomials using Laguerre's method, Department of Computer Science, U. of Toronto.

[Smith 1970] B. T. Smith, Error bounds for zeros of polynomials based upon Gerschgorin's theorems, J. ACM 17, pp. 661-674.

[Stewart 1969] G. W. Stewart, Some iterations for factoring a polynomial, Numer. Math 13, pp. 458-471.

[Traub 1964] J. F. Traub, Iterative Methods for the Solution of Equations, Prentice-Hall.

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

[Witte 1967] Bruno Witte, Comparative tabulations of test problems and test results for mathematical computer routines, SIGNUM Newsletter, Vol. 2, pp. 1-12.

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER
2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER

4. TITLE *(and Subtitle)*
PRINCIPLES FOR TESTING POLYNOMIAL ZEROFINDING PROGRAMS

5. TYPE OF REPORT & PERIOD COVERED

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)
M. A. JENKINS
J. F. TRAUB

8. CONTRACT OR GRANT NUMBER(s)
ONR
N0014-67-A-0314-0010
NR 044-422

9. PERFORMING ORGANIZATION NAME AND ADDRESS
Carnegie-Mellon University
Department of Computer Science
Pittsburgh, Pennsylvania 15213

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

11. CONTROLLING OFFICE NAME AND ADDRESS
Mathematics Program

12. REPORT DATE
March, 1974

13. NUMBER OF PAGES
26

14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)*
Office of Naval Research
Washington, D. C. 20360

15. SECURITY CLASS. *(of this report)*
unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*
approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

The state-of-the-art in polynomial zerofinding algorithms and programs is briefly summarized. The testing of such a program is viewed as requiring four steps: (1) testing program robustness, (2) testing for convergence difficulties, (3) testing for specific weakness of the algorithms, (4) assessment of program performance by statistical testing.

over

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

20. continued:

The authors' algorithm for real polynomials has been reprogrammed with substantial modifications to the decision making processes. In order to verify that the new program is a reliable and efficient mathematical software product, it underwent the testing outlined above. The results of part of this testing are described in this paper.