March 30, 2004

A Robust Strategy for Finding All Real and Complex Roots of Real Polynomials

by C. Bond, ©2003

http://www.crbond.com

Abstract

This paper presents a new strategy for extracting real and complex roots of a real polynomial. The method directly addresses two of the major difficulties in root finding: one, the presence of roots of multiplicity greater than unity and two, the requirement for accurate root estimates to initialize iterative solvers. The strategy leads immediately to the development of a robust *front end* for an iterative solver which may be one of several previously published methods, including Newton's, Bairstow's, Muller's, etc.

1 Background

The roots of polynomials are of interest to more than just mathematicians. They also play a central role in applied sciences including mechanical and electrical engineering where they are used in solving a variety of design problems. Analysis of circuits and systems which involve damping, filtering, resonance, impulse response, and so on often depends in some way on the construction of polynomials and the extraction of their roots. The literature devoted to finding roots of polynomials is extensive and testifies to the practical importance of the subject.

It is known that arbitrary polynomials of order 5 and above cannot be explicitly solved by simple algebraic formulae, and it is because of this that the need for solutions motivates the study of numerical methods. Indeed, the branch of mathematics known as numerical analysis owes much to the contributions and discoveries made in the search for accurate polynomial root finding methods.

2 Polynomial Properties and Notation

The class of polynomials addressed in this paper include all polynomials in a single variable with real coefficients. For convenience, and without loss of generality, we only deal with monic polynomials which can be simply obtained by dividing all coefficients of a given polynomial by the coefficient of the term with highest order. It is easily verified that the roots of the original polynomial and the transformed monic polynomial are identical.

A general polynomial in x can be symbolically represented by a power series in the following form,

$$P = P^n(x) = x^n + a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1x + a_0, \tag{1}$$

where n is the exponent of the term with the highest power and is called the order of the polynomial. It is necessary that the constant term, a_0, differ from zero. Otherwise the polynomial contains the variable x in all terms and, hence, zero is automatically a root. This root may be removed from the problem space by dividing the polynomial by x recursively until the constant term is nonzero. This operation is exact, because it only requires reducing the exponents by one. The other coefficients are not constrained may have any real value, including zero.

Another representation, expressing the polynomial as a product of factors is,

$$P^n(x) = (x - r_1)^a(x - r_2)^b \cdots (x - r_j)^s(x - r_k)^t, \tag{2}$$

where the r_m are roots of the polynomial and the superscripts $a, b \ldots s, t$ are exponents which may have integer values of unity or greater. We will sometimes refer to the exponent of a factor as the multiplicity of the associated root. Note that an arbitrary polynomial may have any combination of single, double or higher power roots, but the requirement for real valued coefficients means that complex roots must occur in conjugate pairs. Hence, if there is a root represented by the factor $x - (\alpha + j\beta)$, then there will also be a root with the factor $x - (\alpha - j\beta)$.

For the polynomial in (2) the sum of all exponents is equal to n.

In a very concrete sense, the task of root finding is to transform the polynomial given as (1) into the form (2) where the roots are explicitly represented.

3 Multiple Roots

Polynomials with roots of multiplicity greater than one, often called repeated roots or multiple roots, present special problems to numerical solvers. The reason for this deserves some discussion, as these problems are not entirely soluble in a finite precision environment and the best we can hope for is to not make them worse.

The major problem which degrades the performance of iterative solvers follows from their typical dependence on the slope of the polynomial in the neighborhood of a zero. Newton's method, for example, computes correction terms for estimated roots from the derivative at the current approximation to a root. Unfortunately, as will be explained in the next section, the derivative becomes zero at a multiple root and stays very close to zero in the immediate neighborhood.

Appealing to bisection or search strategies does not solve the multiple root problem either. This is because when the slope becomes very close to zero near a root, unacceptably large neighborhoods of the root may be indistinguishable from zero, rendering bisection error metrics unstable. The higher the multiplicity of the root, the worse the problem.

3.1 Differentiation of Polynomials

The differentiation of a polynomial has two significant effects which can be exploited in root finding. First, differentiation reduces the order of all multiple roots by one. Second, the roots with unit order do not appear in the derivative polynomial. Hence, if there are no roots of multiplicity greater than one in the original polynomial, the derivative polynomial will have no roots in common with it. On the other hand, double, triple, or higher order roots will survive the differentiation and appear in the derivative polynomial, but with lower order.

To illustrate, we differentiate a simple cubic polynomial which has a pair of identical roots and a single distinct root. The power series form of the cubic, $x^3 + a_1x^2 + a_2x + a_3$, will have a derivative polynomial of 2nd order whose

monic form is $x^2 + b_1 x + b_0$.[1]

Putting the cubic in product form, $(x - r_1)^2(x - r_2)$ and differentiating, we have

$$\begin{aligned} (x - r_1)^2 + 2(x - r_1)(x - r_2) &= (x - r_1)\left((x - r_1) + 2(x - r_2)\right) \quad (3) \\ &= (x - r_1)(3x - r_1 - 2r_2). \quad (4) \end{aligned}$$

Note that the factor $(x - r_1)$ is present in the derivative as well as in the original polynomial and therefore represents a common factor. Notice also that the factor of unit order $(x - r_2)$ is *not* present in the derivative.

The appearance of a root factor of the original polynomial in its derivative clearly highlights why and how multiple roots confound iterative solvers. Near a zero of the polynomial, the derivative also becomes zero. For the naïve Newton's method, the correction term $f(x)/f'(x)$ approaches $0/0$ and becomes numerically unstable.

3.2 Euclid's Algorithm

Although the presence of multiple roots has been shown to cause convergence problems with iterative solvers, it also suggests a solution from outside the traditional domain of numerical analysis. Since multiple roots are associated with the presence of common factors in the original polynomial and its derivative, a decomposition which recovers common factors could lead to solution strategy.

Such a decomposition is possible using Euclid's algorithm.[2] The method used is to divide the original polynomial by its (monic) derivative and test the remainder for zero, indicating the presence of common factors. This operation is numerically stable and can generally be done to machine precision.

[1] Note that the coefficient b_0 may be zero, in which case the derivative polynomial has a root at zero.

[2] Euclid's algorithm is used to determine the *greatest common divisor* (GCD), sometimes called the *greatest common factor* (GCF), of two polynomials.

It is worth noting that the idea of using Euclid's algorithm in the recovery of multiple roots is not without critics. For symbolic or exact calculations no special difficulties are evident. But in a finite precision environment, since numbers may only be approximately represented, the results of division are subject to error. In addition, the original polynomial coefficients may be only approximate. Hence, it is argued that Euclid's algorithm may be no better at isolating multiple roots than an iterative solver. Nevertheless, the author finds that slope or position driven iterative methods, as discussed earlier, suffer from serious problems with error metrics in the neighborhood of multiple roots and that Euclid's algorithm has no such problem. Furthermore, if the algorithm fails to identify multiple roots for whatever reason, the polynomial simply passes through to the solver, none the worse for wear. The performance of the author's implementation of the method presented in this paper consistently exceeds that of the Jenkins-Traub method.[3]

Pseudo-code for the process is shown below.

begin
  $a \leftarrow P$
  $b \leftarrow P'$
  $m \leftarrow n$
  do
    $(q, r) \leftarrow a/b$
    if $r = 0$, **break**
    $a \leftarrow b, b \leftarrow r$
    $m = m - 1$
  while $(m <> 0)$
end

Polynomial Common Factor Algorithm

In this code, P is the original polynomial, n is its order, P' is the monic form of its derivative and (q, r) respresents the quotient and remainder after division. If the algorithm terminates with $r = 0$ the polynomial representing the greatest common factor of the original polynomial and its derivative is

[3]Currently considered the standard for extracting the roots of polynomials.

in b. Otherwise there is no common factor. [4]

For example, if the original polynomial can be written,

$$P = P^7(x) = (x - r_1)^3(x - r_2)^2(x - r_3)(x - r_4) \tag{5}$$

then a (monic) version of the derivative can be written,

$$P' = (x - r_1)^2(x - r_2)(x - r_a)(x - r_b)(x - r_c) \tag{6}$$

where the expression $(x - r_1)^2(x - r_2)$ is the greatest common factor and would be the (monic) version of the polynomial returned in b above.

Now, as the previous example shows, the returned factor may also contain multiple roots. Hence, finding the roots of b may still be difficult, although easier than the original. However, since b is a factor of P, we can divide P by b, with zero remainder, and recover the quotient q. Clearly, $P = q \cdot b$, and we have succeeded in finding a decomposition of P which isolates the distinct roots in q, from their higher order multiples in b.

Following the decomposition it is possible to recover roots from the quotient polynomial without the handicap of dealing with multiple roots. The remaining polynomial b can then be submitted to the same process recursively until all roots of the original polynomial are found.

Although the divisions are subject to the typical accumulation of floating point errors, polynomial differentiation is a stable operation involving simple multiplication by small integers. The net effect of applying this strategy is that the serious difficulties with multiple roots are avoided with only a very small penalty.

4 Root Estimates

A second difficulty encountered in iterative solvers arises from the need for reasonable estimates of the roots to begin the iteration. It is known that

[4] There are some details related to keeping track of the order of b not shown in the pseudo-code.

convergence to some root, in general, can only be assured if the current estimate is 'sufficiently' close to the desired root. Various methods for obtaining such estimates have been published and have found their way into existing solvers. One tactic is to assume zero as a starting estimate in the hope that the solver will converge to the smallest root. Another is to devise an approximation to the smallest root based on the coefficients of the lowest order terms. Attention is often directed to the smallest root because deflation of the polynomial by a small root in preparation for subsequent root extraction inflicts less damage on larger roots. In a sense, this is a damage control strategy.

Root estimates can also be obtained by invoking a global root finder which allows the recovery of all roots simultaneously. Graeffe's method is one example of such a method.

We have chosen the Quotient-Difference (QD) algorithm of Rutishauser as part of the front end of a root finder because it converges to all roots without the need for starting estimates. The QD algorithm is applied iteratively until reasonable approximations to the roots are available. Certain difficulties need to be overcome for use in a general solver, and these are discussed in the next section.

4.1 The Quotient-Difference Algorithm

The QD algorithm is an extension of Bernoulli's method, with the advantage that it converges to all roots rather than just the dominant root. In spite of this virtue, it is not desirable to attempt to find all roots of a polynomial with high precision using the QD algorithm because it is a feed-forward method which often converges slowly and is subject to the accumulation of rounding errors. It can be formulated to serve as a viable root estimator, however, and it will be used as such in the following development.

Some of the issues to resolve in implementing a robust, general purpose root estimator using the QD algorithm are:

- Management of polynomials with one or more zero coefficients,

- Identification of complex roots,
- Algorithm termination criteria.

In addition to these considerations, it is desirable to be able to group root estimates as quadratic factors so that solvers such as Bairstow's method can be employed. Solvers which operate on a single root at a time can still be used by extracting pairs of estimates from the quadratics and refining them successively with the solver.

4.1.1 Root Shifting

A stable formulation of the QD algorithm[5] is initialized from the coefficients of the given polynomial. Unfortunately, if any of the coefficients of the polynomial are zero, the method fails. Hence, it is necessary to be able to transform polynomials in such a way as to alter the coefficients but still be able to recover the original roots.

One such transformation is a simple linear shift of the roots. This has the effect of altering the coefficients and only requires that the estimated roots which are recovered be shifted back by the same amount to undo the change. A simple algorithm exists to transform polynomials in this manner, and with an appropriate choice of shift values it is possible to assure that the transformed polynomial is unlikely to fail.

The root estimation process consists of transforming the original polynomial by shifting the roots, obtaining estimates of the quadratic factors of this polynomial using the QD algorithm, and shifting the estimates obtained back to their appropriate locations.

Complex roots are identified by the behavior of critical values in the QD tableau. These roots, as might be expected, occur in conjugate pairs and will be adjacent in the emerging root estimate list. It is not difficult to recover a quadratic factor from the associated values. Unfortunately, the development

[5] A suitable version is given in "Elements of Numerical Analysis", P. Henrici, John Wiley & Sons, 1964, pp.162-179.

of quadratic factors from the root list is complicated by the arbitrary location of conjugate pairs, which may occur anywhere. This problem can be managed by building a list of quadratic factor estimates from the QD tableau by identifying and extracting the conjugate pairs first and then collecting real estimates in pairs from the remaining values.

5 Summary and Conclusion

Using the methods developed in the preceding exposition, we can now provide an overview of the improved root finding strategy. The overall method separates the solution process into a front end, which removes roots of multiplicity greater than one and then generates reasonable estimates of the remaining (quadratic) factors, and a back end which consists of an iterative root finder. This paper discloses a new strategy for implementing a viable and robust front end.

The front end is recursive, in the sense that polynomials consisting of products of the multiple root terms are recovered with successively reduced order until the multiplicity is reduced to unity. The reduction process exposes multiple roots one by one, so they can be divided out of the original polynomial and added to an expanding *root list* which on conclusion of the process contains all roots. As polynomial factors free of multiple roots are developed, they are submitted to the estimator for determination of estimates of quadratic factors which are used in the back end solver to recover acceptable root values. These are also added to the root list.

A description of the process can now be summarized as follows,

1. Copy the original (monic) polynomial to p_a, $p_a \leftarrow P$,
2. Copy polynomial (monic) derivative to p_b, $p_b \leftarrow p'_a$,
3. Find GCF of p_a/p_b using Euclid's algorithm, $(q, r) = p_a/p_b$,
4. If GCF is 1, submit p_a to estimator/solver, add roots to root list, goto 11
5. Divide GCF out of p_a, $q = p_a/r$,

6. If order of q is less than 3, add roots to root list, goto 10
7. Submit q to estimator/solver, add roots to root list,
8. If order of r less than 3, add roots to roots list, goto 10
9. Copy r to p_a,
10. If more roots to find, goto 2
11. End

The solution process terminates with all roots found. Note that multiple roots are extracted to machine precision by this method, and it is never necessary for the iterative solver to deal with them.

The strategy described leaves untouched a problem which, as previously mentioned, is intrinsically unsolvable in a finite precision environment. It is always possible to construct a polynomial which includes roots which are distinct, but whose difference is so small that that they are indistinguishable in finite precision comparisons. Floating point math platforms can only approximate continuous functions with discrete, but small, steps. So even when exact numbers are expressible in floating point form, simple operations can cause a loss of precision which is not completely recoverable. Furthermore, if the derivative of a polynomial in the neighborhood of a root or group of roots is sufficiently close to zero, a change in the argument may produce an even smaller change in the value of the function, making the construction of an error metric difficult. Hence, in some pathological cases closely spaced roots may be mistakenly identified as identical. In practice it turns out that, because of roundoff errors and loss of precision, the possibility of mistaking identical roots of high order for distinct roots is just as likely.

Nevertheless, and in spite of the limitations of finite precision floating point arithmetic, the root finding strategy above has been shown to provide superior performance in the presence of multiple roots and has been implemented and tested on a wide class of problem polynomials with excellent results.

5.1 Enhancements

Numerical analysts have devised a number of techniques for stabilizing polynomials in the presence of certain catastrophic failure modes. One of the most important of these is root scaling. Scaling is recommended whenever there is a large disparity in the magnitude of the roots of a polynomial, a condition which can be detected from properties of the coefficients. It is strongly recommended that any implementation of the methods described in this paper include a pre-scaler. See the literature for detailed information.