A Hybrid Method for High Precision Calculation of Polynomial Real Roots

George E. Collins* Werner Krandick†
Research Institute for Symbolic Computation
Johannes Kepler University
A-4040 Linz, Austria
e-mail:
gcollins@risc.uni-linz.ac.at
krandick@risc.uni-linz.ac.at

Abstract

A straightforward implementation of Newton's method for polynomial real root calculation using exact arithmetic is inefficient. In each step the length of the iterate multiplies by the degree of the polynomial while its accuracy merely doubles.

We present an exact algorithm which keeps the length of each iterate proportional to its accuracy. The resulting speed-up is dramatic.

The average computing time can be further reduced by trying floating point computations. Several floating point Newton steps are executed; interval arithmetic is used to check whether the result is sufficiently close to the root; if this condition cannot be verified the exact algorithm is invoked.

1 Introduction

Real roots of a univariate integral polynomial A can be calculated in two steps, called "root isolation" and "root refinement". Root isolation computes *isolating intervals* for the roots of A; those are intervals which contain exactly one polynomial root each. Root refinement refines an isolating interval I to the desired width. The roots can be isolated using the coefficient-sign variation method recommended in [5]. This method assumes that A is squarefree; if it is not, polynomial *gcd* calculations can be used to factorize A into squarefree factors. See [1] for a scheme which tries to avoid squarefree factorization.

The root refinement methods presented here will not only assume that the root contained in I is simple, but also that I does not contain any roots of A' or A''. This, again, can be achieved using polynomial *gcd* computations, and, again, one can devise a scheme which tries to avoid them.

Three methods for root refinement will be discussed. Two of the algorithms use exact calculations which produce a subinterval for the root at each Newton step; the third method is a heuristic which uses multiple precision floating point arithmetic to perform a sequence of Newton iterations on the midpoint of I, and then guesses a subinterval for the root.

It will be proved that the straightforward first method described in Section 2 converges quadratically. However, the method is impractical because in each step the length of the interval endpoints multiplies by the degree of the polynomial while the accuracy merely doubles.

The method is therefore modified to truncate the interval endpoints in each step so that they their length becomes appropriate for the width of the subinterval. This implies that the subinterval is slightly enlarged. It is not obvious that the enlarged subinterval will still be contained in the original interval. Section 3 supplies a proof which guarantees containment.

The endpoints of the subinterval will be *binary rational numbers*; these are rational numbers whose denominators are powers of 2. Arithmetic on these numbers is faster than arithmetic on other rational numbers. Section 3 presents an empirical comparison with the method described in Section 2. The speed-up is dramatic.

Section 4 presents preliminary results showing that

*Supported by the Austrian Science Foundation (Grant P8572-PHY).

†Supported by the Austrian Science Foundation (Grant M022-PHY).

$J \leftarrow$ *RATIONAL-NEWTON*(A, I, s, k)

Rational Newton method.

Input : A is a univariate integral polynomial of positive degree. $I = (a_1, a_2)$ is an open interval containing a unique root α of A and containing no roots of A' or A''. $\min(|A'(a_1)|, |A'(a_2)|) \geq \frac{3}{4}\max(|A'(a_1)|, |A'(a_2)|)$. s is the sign of $A'A''$ on I. k is an integer.

Output: J is a subinterval of I of width 2^k or less containing α.

(1) [Initialize.]
$A' \leftarrow$ derivative of A; $J \leftarrow I$;

(2) [Refine interval.]
while $J = (a_1, a_2)$ wider than 2^k do

$$b \leftarrow \begin{cases} A'(a_1) & \text{if } s < 0 \\ A'(a_2) & \text{if } s > 0 \end{cases}$$

$$b_1 \leftarrow a_1 - \frac{A(a_1)}{b}$$

$$b_2 \leftarrow a_2 - \frac{A(a_2)}{b}$$

$$J \leftarrow (b_1, b_2)$$

(3) [Return.]
return(J);

Figure 1: Algorithm 1

a further impressive speed-up can be obtained by using floating point arithmetic. Interval arithmetic is used for *a posteriori* verification of the floating point result. If verification fails the exact method of Section 3 is used as a back-up; the combined method is totally correct and fast on the average.

Section 5, finally, points out questions which deserve further study.

The algorithms were implemented in Portable C using the `saclib`-library [2] of C-callable computer algebra programs. Timing results were obtained running optimized code on a DECstation 5000/200 under ULTRIX V4.0.

2 Rational Calculation

An isolating interval $I = (a_1, a_2)$ for a simple real root of a polynomial A can be refined iteratively using Algorithm 1 in Figure 1. As mentioned in Section 1 it may be assumed that neither A' nor A'' have roots in I, hence the sign of $A'A''$ on I is well-defined. Depending on *sign*$(A'A'')$ on I the algorithm chooses one of the points $(a_i, A(a_i))$, $i = 1, 2$, for constructing a tangent to the graph of A. It then constructs the parallel to this tangent through the other point. The points where the two straight lines intersect with the x-axis are the endpoints of the interval J. Proposition 1 shows that J will indeed be a subinterval of I, and that its width will be at most a quarter of the width of I — provided the interval I was small enough to begin with. Moreover, the latter statement will hold for all pairs of iterated intervals. This property will be used in proving the correctness of Algorithm 2 in Section 3.

Proposition 1 *Let A be a univariate integral polynomial of positive degree. Let $I = (a_1, a_2)$ be an open interval containing a simple root α of A but no roots of A' and A''. Let $s \neq 0$ be the sign of $A' \cdot A''$ on I, and assume*

$$\min(|A'(a_1)|, |A'(a_2)|) \geq \frac{3}{4}\max(|A'(a_1)|, |A'(a_2)|).$$

Define

$$b = \begin{cases} A'(a_1) & \text{if } s < 0 \\ A'(a_2) & \text{if } s > 0 \end{cases}$$

Now let

$$b_1 = a_1 - \frac{A(a_1)}{b}$$

$$b_2 = a_2 - \frac{A(a_2)}{b}$$

Then

1. $a_1 < b_1 < \alpha < b_2 < a_2$,
2. $b_2 - b_1 < (a_2 - a_1)/4$ *and*
3. $\min(|A'(b_1)|, |A'(b_2)|) \geq \frac{3}{4}\max(|A'(b_1)|, |A'(b_2)|)$.

Proof. The proposition is proved for the case $s < 0$, *sign*$(A') > 0$, *sign*$(A'') < 0$; the other cases can be proved similarly.

1. According to the mean value theorem

$$\frac{A(\alpha) - A(a_1)}{\alpha - a_1} = A'(\xi)$$

for some $\xi \in (a_1, \alpha)$.

This implies $0 = A(\alpha) > A(a_1)$ since $A'(\xi) > 0$ and $\alpha - a_1 > 0$; hence, $a_1 < b_1$ follows from *sign*$(A') > 0$ and the definition of b_1.

Furthermore, $A'(\xi) < A'(a_1)$ since *sign*$(A'') < 0$, and so

$$\frac{-A(a_1)}{\alpha - a_1} < A'(a_1),$$

i.e. $-A(a_1) < A'(a_1)(\alpha - a_1)$. So, because of $A'(a_1) > 0$

$$b_1 = a_1 - \frac{A(a_1)}{A'(a_1)} < \alpha.$$

Analogously one obtains $A(a_2) > 0$, $\alpha < b_2$ and $b_2 < a_2$.

2. Since $sign(A') > 0$ $|A'(a_1)| = A'(a_1)$ and $|A'(a_2)| = A'(a_2)$. Now $sign(A'') < 0$ implies $A'(a_1) > A'(a_2)$. Therefore $\min(|A'(a_1)|, |A'(a_2)|) = A'(a_2)$, $\max(|A'(a_1)|, |A'(a_2)|) = A'(a_1)$, and so the "3/4-assumption" is equivalent to

$$A'(a_2) \geq \frac{3}{4}A'(a_1).$$

The mean value theorem combined with the fact that the second derivative is negative on I now renders

$$\frac{3}{4}A'(a_1) \leq A'(a_2) < \frac{A(a_2) - A(a_1)}{a_2 - a_1},$$

i.e.

$$\frac{3}{4}(a_2 - a_1) < \frac{A(a_2) - A(a_1)}{A'(a_1)}.$$

Therefore

$$b_2 - b_1 = a_2 - a_1 - \frac{A(a_2) - A(a_1)}{A'(a_1)} < \frac{1}{4}(a_2 - a_1).$$

3. Since A' is monotone decreasing $a_1 < b_1$ implies

$$\frac{3}{4}A'(a_1) > \frac{3}{4}A'(b_1).$$

As above,

$$A'(a_2) \geq \frac{3}{4}A'(a_1).$$

Now $b_2 < a_2$ implies $A'(b_2) > A'(a_2)$, so,

$$A'(b_2) > A'(a_2) \geq \frac{3}{4}A'(a_1) > \frac{3}{4}A'(b_1),$$

and so

$$\min(|A'(b_1)|, |A'(b_2)|) \geq \frac{3}{4}\max(|A'(b_1)|, |A'(b_2)|)$$

(using $sign(A') > 0$ and $sign(A'') < 0$) □

Proposition 2 *Algorithm 1 converges quadratically.*

Proof. Again, the proposition is proved for the case $s < 0$, $sign(A') > 0$, $sign(A'') < 0$; the other cases can be proved similarly.

Let

$$J_1 \supset J_2 \supset \ldots$$

be the sequence of intervals $J_j = (a_{1,j}, a_{2,j})$ computed by Algorithm 1. Let $w_j = a_{2,j} - a_{1,j}$ be the width of the interval J_j. It will be shown that the sequence (w_j) converges quadratically to zero.

The w_j can be written as

$$w_j = |\gamma_j| + \delta_j + \epsilon_j,$$

with $\gamma_j = a_{1,j} - \alpha$, $\delta_j = \hat{a}_j - \alpha$ and $\epsilon_j = a_{2,j} - \hat{a}_j$, where

$$\hat{a}_j = a_{2,j-1} - \frac{A(a_{2,j-1})}{A'(\alpha)}.$$

It will be shown that each of the sequences $(|\gamma_j|)$, (δ_j) and (ϵ_j) converges quadratically to zero.

Sequence $(|\gamma_j|)$ converges quadratically to zero; this is just the quadratic convergence of ordinary Newton's method.

Sequence (δ_j) converges quadratically to zero; this is the quadratic convergence of a special case of Whittaker's method. For a proof see [3].

Sequence (ϵ_j), finally, converges quadratically to zero because sequence $(|\gamma_j|)$ does so. Indeed,

$$\begin{aligned} \epsilon_j &= -\frac{A(a_{2,j-1})}{A'(a_{1,j-1})} + \frac{A(a_{2,j-1})}{A'(\alpha)} \\ &= (A'(a_{1,j-1}) - A'(\alpha))\frac{A(a_{2,j-1})}{A'(a_{1,j-1})A'(\alpha)}. \end{aligned}$$

The fraction on the right can be bounded by a constant C_1. The remaining part of ϵ_j can be bounded as follows

$$\begin{aligned} (A'(a_{1,j-1}) - A'(\alpha)) &= A'(\alpha + \gamma_{j-1}) - A'(\alpha) \\ &= \textstyle\sum_{m=1}^{\infty} \gamma_{j-1}^{m} \frac{A^{(m+1)}(\alpha)}{m!} \\ &= \gamma_{j-1} \textstyle\sum_{m=1}^{\infty} \gamma_{j-1}^{m-1} \frac{A^{(m+1)}(\alpha)}{m!} \\ &\leq |\gamma_{j-1}|C_2, \end{aligned}$$

where C_2 is a constant. Thus, $\epsilon_j \leq |\gamma_{j-1}|C_2C_1$, and (ϵ_j) converges quadratically to zero because $(|\gamma_j|)$ does so. □

Table 1 presents tracing results of a typical computation of Algorithm 1 for a root of a polynomial of degree 10 with randomly generated 10-bit coefficients. Snapshots taken before each Newton step show $|\lfloor \log_2(w) \rfloor|$, where w is the width of the isolating interval, and the bit length of the denominator of its left endpoint. The data were obtained when refining the isolating interval $I = (11/32, 353/1024)$ for a root of the polynomial

Table 1: Growth of Rational Numbers

Rational Newton steps, $deg(A) = 10$	
bits of accuracy	bits in denominator
10	6
20	101
39	1081
77	10914

$A(x) = 914x^{10} - 993x^9 + 3x^8 - 505x^7 + 415x^6 - 916x^5 + 592x^4 - 243x^3 - 788x^2 - 81x + 127$. The sequence of accuracies agrees well with the quadratic convergence of the algorithm. The length of the rational numbers roughly multiplies by $deg(A)$ in each step thus making the algorithm impractical.

3 Binary Rational Calculation

Proposition 1 can be strengthened to take advantage of binary rational arithmetic and to keep the denominators of the binary rational numbers in proportion with the width of the subinterval computed in each step.

The idea for the improvement is to take the interval $J = (b_1, b_2)$ which is obtained from one Newton step and to expand it in such a way that the endpoints of the expanding interval snap to points of a *grid*

$$G_{2^h} = 2^h \cdot \mathbf{Z},$$

$h \in \mathbf{Z}$, whose points are binary rational numbers and whose mesh size 2^h is proportional to the width of J.

If the endpoints of $I = (a_1, a_2)$ are binary rationals which are almost consecutive integral multiples of the same power of 2, and if G_{2^h} is fine enough to contain a_1 and a_2, then the expanded interval $K = (c_1, c_2)$ will be a subinterval of I.

Proposition 3 shows how such a construction can be carried out in a way suitable for iteration. Figure 2 shows the resulting Algorithm 2.

Proposition 3 *In the situation of Proposition 1 assume in addition that*

$$a_1 = m_1 2^k,$$

$$a_2 = m_2 2^k$$

with integers k, m_1, m_2 and

$$m_2 - m_1 \leq 5.$$

Define c_1 and c_2 as follows:

$$h = \lfloor \log(b_2 - b_1) \rfloor - 1$$

$$t = 2^h$$

$J \leftarrow$*BINARY-RATIONAL-NEWTON*(A, I, s, k)

Binary rational Newton method.

Input : A is a univariate integral polynomial of positive degree. $I = (a_1, a_2)$ is an open interval containing a unique root α of A and containing no roots of A' or A''. $\min(|A'(a_1)|, |A'(a_2)|) \geq \frac{3}{4}\max(|A'(a_1)|, |A'(a_2)|)$. Furthermore, $a_1 = m_1 2^e$ and $a_2 = m_2 2^e$ with integers m_1, m_2, e and such that $m_2 - m_1 \leq 5$. s is the sign of $A'A''$ on I. k is an integer.

Output: $J = (c_1, c_2)$ is a subinterval of I of width 2^k or less containing α. Furthermore, $c_1 = n_1 2^t$ and $a_2 = n_2 2^t$ with integers n_1, n_2, t and such that $n_2 - n_1 \leq 5$.

(1) [Initialize.]
$A' \leftarrow$ derivative of A; $J \leftarrow I$;

(2) [Refine interval.]
while $J = (a_1, a_2)$ wider than 2^k do

$$b \leftarrow \begin{cases} A'(a_1) & \text{if } s < 0 \\ A'(a_2) & \text{if } s > 0 \end{cases}$$

$$b_1 \leftarrow a_1 - \frac{A(a_1)}{b}$$

$$b_2 \leftarrow a_2 - \frac{A(a_2)}{b}$$

$$h \leftarrow \lfloor \log(b_2 - b_1) \rfloor - 1$$

$$t \leftarrow 2^h$$

$$c_1 \leftarrow \left\lfloor \frac{b_1}{t} \right\rfloor t$$

$$c_2 \leftarrow \left\lceil \frac{b_2}{t} \right\rceil t$$

$$J \leftarrow (c_1, c_2)$$

(3) [Return.]
return(J);

Figure 2: Algorithm 2

$$c_1 = \left\lfloor \frac{b_1}{t} \right\rfloor t$$

$$c_2 = \left\lceil \frac{b_2}{t} \right\rceil t$$

Then

1. $a_1 \le c_1 \le b_1 < \alpha < b_2 \le c_2 \le a_2$,
2. $c_2 - c_1 < (a_2 - a_1)/2$,
3. $\min(|A'(c_1)|, |A'(c_2)|) \ge \frac{3}{4} \max(|A'(c_1)|, |A'(c_2)|)$,
4. *c_1 and c_2 are binary rational numbers, namely integral multiples of t, and $c_2/t - c_1/t \le 5$.*

Proof.

1. Clearly, $c_1 \in G_t$ is the greatest integral multiple of t which is $\le b_1$, and $c_2 \in G_t$ is the smallest integral multiple of t which is $\ge b_2$. So, it suffices to show that a_1 and a_2 are also integral multiples of t.

   According to Proposition 1 $b_2 - b_1 < (a_2 - a_1)/4$. Hence, due to the new condition on a_1 and a_2,

   $$b_2 - b_1 < 5 \cdot 2^{k-2}.$$

   Now

   $$h = \lfloor \log(b_2 - b_1) \rfloor - 1 \le \lfloor \log 5 + k - 2 \rfloor - 1 = k - 1,$$

   hence $t \le 2^{k-1}$ and thus a_1 and a_2 are integral multiples of t.

2. Because of Proposition 1 it suffices to show that $c_2 - c_1 \le 2(b_2 - b_1)$. Indeed, $c_2 - c_1 = (c_2 - b_2) + (b_2 - b_1) + (b_1 - c_1) \le 2t + (b_2 - b_1) \le 2(b_2 - b_1)$.

3. Follows as in the proof of Proposition 1.

4. From $\lfloor \log(b_2 - b_1) \rfloor = h + 1$ it follows that $\log(b_2 - b_1) < h + 2$ and thus,

   $$b_2 - b_1 < 2^{h+2} = 4t.$$

   This result applied to

   $$\frac{c_2}{t} - \frac{c_1}{t} = \frac{c_2 - b_2}{t} + \frac{b_2 - b_1}{t} + \frac{b_1 - c_1}{t} < \frac{b_2 - b_1}{t} + 2$$

   yields

   $$\frac{c_2}{t} - \frac{c_1}{t} < 4 + 2 = 6 \square$$

Table 2 was obtained from the same example as Table 1, however Algorithm 2 was used. The table indicates that Algorithm 2 converges as well as Algorithm 1,

Table 2: Controlled Growth of Binary Rational Numbers

Binary rational Newton steps, $deg(A) = 10$	
bits of accuracy	bits in denominator
10	6
20	22
38	40
75	77
148	148

Table 3: Comparing Algorithms 1 and 2

	from 10 to 40 bits of accuracy		
	deg 5	deg 10	deg 15
Algorithm 1	533 ms.	20332 ms.	108468 ms.
Algorithm 2	33 ms.	83 ms.	100 ms.

and it shows that the length of the binary rational endpoints is kept at about the binary logarithm of the interval width. Consequently, Algorithm 2 is vastly superior to the straightforward Algorithm 1. Table 3 lists average times in milliseconds for refining one isolating interval from width 2^{-10} to width 2^{-40} for each of the algorithms. All polynomials had 10-bit coefficients; the degrees were varied from 5 to 15.

4 Floating Point and Interval Arithmetic Calculation

Algorithm 2 was obtained from Algorithm 1 by truncating the Newton iterates after having computed them exactly. Further speed-up can be obtained by truncating numbers not just after each Newton step but rather after each arithmetic operation. In order to gauge the speed-up potential of floating point arithmetic the following heuristic version of Newton's method was devised.

The polynomial A, its derivative and the midpoint c of the isolating interval I are converted to floating point representation; for A the precision is chosen such that all coefficients can be represented exactly; c is represented with the precision necessary to represent the width of the desired refined interval. The number of required Newton steps is guessed based on the assumption that each Newton step will double the accuracy obtained in the previous step. The Newton steps are executed using multiple precision floating point arithmetic (see [4], [6]). The last iterate is taken to be the midpoint of a subinterval $J \subseteq I$ of the desired width.

The signs of A on the endpoints of J are evaluated to check whether J indeed contains the root; these evaluations are done using interval arithmetic; if inter-

Table 4: Comparing Algorithm 2 and Heuristic

	from 10 to 100 bits of accuracy				
	deg 10	deg 20	deg 30	deg 40	deg 50
Algorithm 2	.17s	.44s	1.44s	1.88s	2.71s
Heuristic	.04s	.08s	.12s	.15s	.18s
	from 10 to 1000 bits of accuracy				
	deg 10	deg 20	deg 30	deg 40	deg 50
Algorithm 2	8.7s	30.4s	63.7s	99.9s	158.9s
Heuristic	.7s	1.1s	1.7s	2.2s	2.7s

val arithmetic fails, binary rational arithmetic is used. If J does not contain the root, J is discarded and the interval I is refined using Algorithm 2.

Table 4 compares computing times of the heuristic and Algorithm 2. Listed are averages of computing times in seconds for refining one isolating interval of width 2^{-10} to width 2^{-100} and to width 2^{-1000}. All input polynomials had randomly generated 10-bit coefficients. The degrees were varied from 10 to 50. A total of 204 interval refinements were computed for the table; in one case the heuristic had to fall back to Algorithm 2. For consistency this case was ignored when the average computing time was calculated.

5 Future Work

The heuristic described in Section 4 could be speeded up by starting the floating point iterations with a small precision which would be increased after each Newton step. This should produce a speed-up proportional to the logarithm of the final number of bits of accuracy. It would be useful to investigate what precision should be used in the verification step so that the interval computations will be carried out with no more than appropriate precision.

References

[1] George E. Collins and Werner Krandick. An efficient algorithm for infallible complex root isolation. In *International Symposium on Symbolic and Algebraic Computation*, pages 189–194, ACM Order Department, P.O.Box 64145, Baltimore, MD 21264, Phone: 1-800 342-6626 or 1-410 528-4261, 1992. ACM.

[2] George E. Collins et al. `saclib` user's guide. Technical Report RISC-Linz Report Series Number 93-19, Research Institute for Symbolic Computation, RISC-Linz, Johannes Kepler University, A-4040 Linz, Austria, 1993.

[3] Peter Henrici. *Elements of Numerical Analysis*, John Wiley & Sons, Inc., New York, London, Sydney, 1964.

[4] J. R. Johnson and W. Krandick. A multiprecision floating point and interval arithmetic package for symbolic computation. Technical Report RISC-Linz Report Series Number 93-20, Research Institute for Symbolic Computation, RISC-Linz, Johannes Kepler University, A-4040 Linz, Austria, 1993.

[5] Jeremy R. Johnson. Algorithms for polynomial real root isolation. Technical Research Report OSU-CISRC-8/91-TR21, The Ohio State University, 2036 Neil Avenue Mall, Columbus, Ohio 43210, Phone: 614-292-5813, 1991.

[6] W. Krandick and J. R. Johnson. Efficient multiprecision floating point multiplication with optimal directional rounding. In *11th IEEE Symposium on Computer Arithmetic*, IEEE Computer Society Press, P.O.Box 3041, Los Alamitos, CA 90720-1264, Phone: 714 821-8380, 1993.