# An Efficient Algorithm for Infallible Polynomial Complex Root Isolation*

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

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

## Abstract

Applying the principle of the argument to rectangles provides an efficient algorithm for polynomial complex root isolation. For example, the roots of a real integral polynomial of degree 10 (with 10-bit coefficients) can be isolated in about 1 second. Furthermore, although the algorithm is not designed for efficient refinement of isolating rectangles, it does nevertheless refine all of these rectangles to width $10^{-50}$ in less than 4 minutes.

## 1 Introduction

Let $A(z)$ be a non-zero squarefree univariate polynomial with Gaussian integer coefficients. This paper presents an algorithm which produces disjoint isolating rectangles for all the roots of $A(z)$ in the complex plane, i.e. each such rectangle will contain exactly one root of $A$, and each root of $A$ will be contained in one of the rectangles. The sides of the rectangles will be parallel to the axes. Given any rational number $e > 0$, the algorithm will deliver isolating rectangles with sidelengths $< e$. Unlike numerical methods the algorithm will always terminate with correct results. Negotiating the computational price of infallibility is the main thrust of this work. Computing time experiments suggest that the presented algorithm can be recommended as an efficient first step in a complex root calculation scheme, namely to provide starting points for a rapidly converging root approximation method. The average computing time of the algorithm seems to be approximately cubic in the degree of $A$ and linear in the length of its coefficients.

The crux of the method is an algorithm which determines the number of roots in a rectangle. Possible additional applications of this algorithm include hybrid numerical-symbolic root calculation schemes, in which the routine can be used to verify numerical computation steps.

Pinkert's work [5], [6], [7] seems to be the first experimental study of an exact method for localizing complex polynomial roots. Based on a theorem by Routh his algorithm uses Sturm sequences to determine for a complex polynomial, how many of its roots lie in the upper half-plane. This information is used together with polynomial transformations, among them root squaring, to determine the number of roots in a rectangle.

Wilf [8] shows how Sturm sequences can be applied to determine the number of roots in a rectangle directly without costly transformations.

A recent study by Johnson [2] on real root isolation shows, however, that just the computation of the Sturm sequence of a polynomial by itself is far more time consuming than isolating all real roots of that polynomial using the coefficient sign variation method. As a result, our method, which is based on the coefficient sign variation method, is far more efficient than an exact implementation of Wilf's method would be.

Methods which do not use Sturm sequences are outlined by Lehmer [3] and Collins [1]. Lehmer's method computes sequences of polynomials somewhat analogous to Sturm sequences, and one might guess that the coefficient growth would be comparable to that for Sturm sequences, and, as a result would be equally time consuming to compute. Furthermore, the use of circles leads to difficulties which do not occur for rectangles.

We use the method described in [1]. However, we corrected its flaws and made some non-trivial improvements: The analysis in Section 2 is sufficiently general to deal with cases which were not considered in [1], namely with the possibility that certain polynomials which arise in the computation either have multiple roots or vanish identically. The main steps of the computation and the basic data structures are described in Sections 3 and 4, respectively, Section 5 compares two bisection strategies with respect to their memory requirements. Section 6 demonstrates that one of the basic computational steps outlined in Section 3 can be avoided for almost all input polynomials without ever compromising infallibility. Section 7 gives a device to speed up the algorithm in case the input polynomial is real.

All experimental results were obtained on a DECstation 5000/200 running ULTRIX V4.0. The algorithms were implemented in Portable C on top of the `saclib`-library of computer algebra programs.

## 2 Mathematical Basis

Given an algorithm for determining the number of roots of $A$ in any rectangle, all the roots of $A$ can be isolated by a bisection process: From the coefficients of $A$ a rational *a priori* root bound $b$ for $A$ is calculated. The root bound defines an initial square centered at the origin and of sidelength $2b$ which is guaranteed to contain all roots. The initial square is then bisected, and the number of roots in each half is determined. A half which does not contain any roots is discarded. A half which contains exactly one root and whose edges have lengths $< e$ is appended to the solution list. A half which contains either more than one root or which contains one root but has an edge of length $\geq e$ is appended to the working list. The bisection process continues for rectangles in the working list. Squares are bisected vertically, non-squares are bisected orthogonally to their longer edges until isolating rectangles of the required size have been found.

The number $r$ of roots of $A$ in a rectangle $R$ is determined using the principle of the argument from complex analysis: If $\partial R$ designates the boundary of $R$, $r$ is equal to the number of times the curve $A(\partial R)$ winds around the origin. For an elementary proof see [4].

The winding number of $A(\partial R)$ is determined from the sequence of quadrants traversed by $A(z)$ as $z$ traverses $\partial R$. Here, "sequence of quadrants" refers to a sequence of *generalized quadrants*, i.e. the four *traditional quadrants*, the two positive and the two negative half-axes, and the origin. The sequence of quadrants is determined by determining the generalized quadrant of $A(z)$ at some starting point $z = z_0 \in \partial R$ and by keeping track of the quadrant changes of $A(z)$ as $z$ traverses $\partial R$. Each quadrant change occurs at an *axial point*, i.e. at a point $z = p \in \partial R$ with $A(p)$ lying on the real or the imaginary axis.

The following will show that there are only a finite number of axial points at which a quadrant change can occur. The boundary of $R$ consists of four *edges* — open line segments each of which is parallel to one of the coordinate axes — and the four corner points. If $E$ is an edge of $R$, its image $A(E)$ lies either entirely on one of the coordinate axes or its intersection with either axis is finite. In the first case $E$ is called an *axial edge*, and the only points $p$ of $E$ at which a quadrant change of $A(z)$ can occur are the finitely many roots of $A$ on $E$. This motivates the definition of the *set of critical points* of a rectangle $R$ to be the set of axial points on $\partial R$ minus the axial edges of $R$ but including the roots of $A$ on $\partial R$. To summarize, quadrant changes may occur at the critical points of $\partial R$, the number of critical points is finite, the sequence of quadrants for $A(\partial R)$ is finite, and the problem of determining the sequence of quadrants is reduced to localizing and classifying the critical points.

Localizing critical points on an edge means giving disjoint open line segments or one-point line segments on the edge, each of which contains exactly one critical point. A critical point on a corner is localized by giving its coordinates.

Classification of a critical point $p$ provides information about the behavior of $A(z)$ as $z$ traverses the neighborhood of $p$ on $\partial R$. Here is a complete listing of the possible instances of this behavior:

1. If $p$ is on a non-axial edge, $A(z)$ can
   - touch an axis (without crossing it) — not at the origin.
   - cross an axis — not at the origin.
   - cross both axes at the origin.
   - cross one axis and not the other at the origin.
2. If $p$ is on an axial edge, $A(p) = 0$ and $A(z)$ goes from one half of the axis to its other half.
3. If $p$ is a cornerpoint of the rectangle, the behavior of $A(z)$ depends on whether the edges incident in $p$ are both non-axial or both axial, or whether one is axial and the other non-axial.
   - If both edges are non-axial $A(z)$ can behave as listed in case 1.
   - If both edges are axial $A(z)$ either
     - stays on the same half axis or
     - changes from one half of an axis into its other half or
     - turns right from one half axis into one half of the other axis or
     - turns left from one half axis into one half of the other axis.
   - If one edge is axial and the other edge is non-axial, $A(z)$ can

– go from one traditional quadrant into an adjacent half-axis (or conversely) or
– go from one traditional quadrant into a non-adjacent half-axis (or conversely), necessarily crossing the origin.

Once the sequence of (generalized) quadrants for $A(\partial R)$ is determined, the winding number of $A(\partial R)$ can be calculated by accumulating the net argument change of $A(\partial R)$ in units of $\pi/4$ and dividing the total by $2\pi$.

If $z$ traverses $\partial R$ in counterclockwise direction, each quadrant change of $A(z)$ in counterclockwise direction contributes positively to the net argument change, while each quadrant change of $A(z)$ in clockwise direction contributes negatively to the net argument change.

A quadrant change from one quadrant into the same quadrant contributes an argument change of 0. A quadrant change from one traditional quadrant into an adjacent traditional quadrant by crossing an axis — not at the origin — contributes an argument change of $\pm\pi/2$.

Quadrant changes which involve a crossing of the origin are ambiguously counterclockwise or clockwise. The ambiguity is resolved by deciding whether or not to count roots of $A$ on $\partial R$: According to the principle of the argument the net argument change obtained by counting such a root is greater by $2\pi$ than the net argument change obtained by not counting the root.

In the context of the bisection scheme it is important that roots which lie on the boundaries of several adjacent rectangles be counted with exactly one rectangle. This condition is met by stipulating for instance that roots of $A$ on the north edge, the east edge and on the north-east corner of any rectangle $R$ shall be counted, while roots on the other parts of $\partial R$ shall not be counted.

A quadrant change from one traditional quadrant into an adjacent traditional quadrant by crossing the origin contributes an ambiguous argument change of $+\pi/2$ or $-3\pi/2$. A quadrant change from a traditional quadrant into the diagonally opposite quadrant contributes an ambiguous argument change of $\pm\pi$. Likewise, a quadrant change from one half-axis into the opposite half-axis contributes an ambiguous argument change of $\pm\pi$. A quadrant change from turning right from one half-axis into one half of the other axis contributes an ambiguous argument change of $+\pi/2$ or $-3\pi/2$. A quadrant change from turning left from one half-axis into one half of the other axis contributes an ambiguous argument change of $-\pi/2$ or $+3\pi/2$. A quadrant change from a traditional quadrant into the half-axis which is adjacent in counterclockwise direction contributes an ambiguous argument change of $+9\pi/4$ or $+\pi/4$; the converse quadrant change contributes an ambiguous argument change of $+7\pi/4$ or $-\pi/4$. Likewise, a quadrant change from a traditional quadrant into the half-axis which is adjacent in clockwise direction contributes an ambiguous argument change of $+7\pi/4$ or $-\pi/4$; the converse quadrant change contributes an ambiguous argument change of $+\pi/4$ or $+9\pi/4$. A quadrant change from a traditional quadrant into the non-adjacent half-axis which is encountered first when going in counterclockwise direction contributes an ambiguous argument change of $+3\pi/4$ or $-5\pi/4$; the converse quadrant change contributes an ambiguous argument change of $+5\pi/4$ or $-3\pi/4$. A quadrant change from a traditional quadrant into the non-adjacent half-axis which is encountered second when going in counterclockwise direction contributes an ambiguous argument change of $+5\pi/4$ or $-3\pi/4$; the converse quadrant change contributes an ambiguous argument change of $+3\pi/4$ or $-5\pi/4$.

# 3 Main Parts of the Computation

Critical points on edges of rectangles can be found, classified and localized in the following four computational steps:

**Real Representation of Complex Polynomials** The set of all axial points of $A$ can be represented as the union of the zero sets of two bivariate real polynomials. If $A(z)$ is the input polynomial with Gaussian integer coefficients, the integral polynomials $A_1(x,y)$ and $A_2(x,y)$ with

$$A(x+iy) = A_1(x,y) + iA_2(x,y)$$

are, respectively, called the *real part* and the *imaginary part* of $A$.

**Specialization** The set of axial points of $A$ on a straight vertical line $L = \{s\} \times \mathbf{R}$ is the union of the sets of real zeros of two univariate real polynomials obtained from the real and imaginary parts of $A$ by specializing $x$ to $s$:

$$\tilde{A}_1(y) = A_1(s,y),$$

$$\tilde{A}_2(y) = A_2(s,y).$$

If $\eta$ is a zero of $\tilde{A}_1(y)$, $p = (s,\eta)$ is an axial point $p \in L$ with $A(p)$ on the imaginary axis. An analogous assertion holds for the zeros of $\tilde{A}_2(y)$.

Likewise, the set of axial points on a horizontal line can be represented as the union of the real zeros of two univariate real polynomials by specializing $y$ to $s$ in the real and imaginary parts of $A$.

Table 1: Itemized Computing Times

| coeff. bit-length 10 | | | |
|---|---|---|---|
| degree | real root iso. | real root ref. | specialization |
| 10 | 40 % | 23 % | 20 % |
| 20 | 43 % | 20 % | 28 % |
| 30 | 47 % | 16 % | 31 % |
| 40 | 47 % | 15 % | 34 % |
| 50 | 48 % | 13 % | 34 % |

Table 2: Bisection Strategies

| coeff. bit-length 10 | | |
|---|---|---|
| degree | breadth-first | depth-first |
| 10 | 2668 | 2044 |
| 20 | 12765 | 6862 |
| 30 | 33184 | 15511 |
| 40 | 73825 | 24349 |
| 50 | 136268 | 45551 |

After multiplication by a suitable power of the denominator of $s$ and abuse of notation, the polynomials $\tilde{A}_1$ and $\tilde{A}_2$ are assumed to be integral. These multiplications can be done fast because the rational numbers $s$ which arise in the bisection process as fixed coordinates of bisecting lines are in fact *binary rationals*: A power of 2 is chosen as the root bound $b$ for $A$ ($b$ is used to define the initial square), so that the denominators of the numbers $s$ will always be powers of 2.

If neither of the polynomials $\tilde{A}_1$ and $\tilde{A}_2$ is zero, the critical points of $A$ on $L$ are given by the combined zeros of each polynomial. If one of the polynomials $\tilde{A}_1$ and $\tilde{A}_2$ is the zero polynomial, the other polynomial will be different from zero, and its roots will represent the critical points of $A$ on $L$.

**Squarefree Basis Computation** Critical points $p = (s, \eta) \in L$ will be classified according to the behavior of $\tilde{A}_1$ and $\tilde{A}_2$ at $\eta$, namely according to the multiplicities $m_k$ of $\eta$ in $\tilde{A}_k$, $k = 1, 2$. Here $m_k$ can be 0 if $\tilde{A}_k(\eta) \neq 0$ and $\infty$ if $\tilde{A}_k = 0$. The multiplicities of critical points $p$ are obtained by computing a squarefree basis for $S = \{\tilde{A}_1, \tilde{A}_2\} - \{0\}$ together with the multiplicities of the basis polynomials in the polynomials in $S$. Now the critical points are represented by the set of the combined zeros of squarefree, pairwise prime polynomials.

**Real Root Isolation and Refinement** The critical points on an edge $\{s\} \times I$ or $I \times \{s\}$ are localized in two steps. For each basis polynomial its zeros in the open interval $I$ are isolated using the coefficient-sign variation method. Then isolating intervals which belong to zeros of different polynomials are refined by bisection until they are disjoint.

Table 1 lists average percentages of the total computing time spent on these computational steps when the roots of integral polynomials of various degrees are isolated.

# 4 Basic Data Structures

The subrectangles which result from a vertical or horizontal bisection of a rectangle have each one edge and two half-edges in common with the original rectangle. The critical points on these parts of their boundaries do not need to be localized and classified again — provided the pertaining data is stored with the original rectangle. The basic data structures employed by the algorithm are listed here from the bottom to the top of the hierarchy:

**Line Record** The line record for a line $L = \{s\} \times \mathbf{R}$ or $L = \mathbf{R} \times \{s\}$ is the corresponding set of basis polynomials. This set is obtained by specializing the real representation of the input polynomial, and by computing a squarefree basis of the specialized polynomials.

**Isolation List** The list of isolating segments for the critical points on an edge. This list is obtained from the line record by real root isolation and refinement. The corresponding classification data is also included in the isolation list.

**Edge Record** The isolation list and the line record for the edge. The line record is included because its polynomials will be needed in case one of the isolating segments in the isolation list needs to be refined. This will be required whenever a bisecting line intersects such a segment.

**Rectangular Record** The edge records of the four edges which delineate the rectangle.

# 5 Bisection Strategy

Table 2 compares memory requirements under two bisection strategies:

The *depth-first* strategy always chooses the smallest rectangle on the working list to be bisected. This results in reducing the size of one rectangle while the other rectangles stay large. The *breadth-first* strategy always chooses the largest rectangle on the working list to be bisected. This results in making several passes over the working list, such that after each pass the rectangles in the working list will be congruent.

A disadvantage of the breadth-first strategy is due to the growth of the individual line records during the bisection process: The lengths of the coefficients of the specialized polynomials grow with the lengths of the rational numbers which are used for specializing. Thus the records of small rectangles will require more memory space than the records of large rectangles. By decreasing pass-by-pass the size of all rectangles simultaneously, the breadth-first strategy increases the size of all rectangular records at the same time.

The depth-first strategy, by contrast, lets the record of only one rectangle grow, discards it when the rectangle is appended to the solution list, and then turns to the next rectangle.

Table 2 lists the maximum number of memory cells required for storing rectangular records during each version of the algorithm.

# 6 Avoiding Squarefree Basis Computation

A significant improvement of the algorithm is made possible by a particular feature of the coefficient sign variation method for real root isolation: When the method is applied to isolate a multiple real root of a polynomial, it will simply run forever. This characteristic gives rise to a scheme in which the algorithm for complex root isolation skips the computation of a squarefree basis and first tries to isolate the roots of each specialized polynomial: Only if the roots cannot be isolated within a certain number of bisection steps (of the coefficient sign variation method), will the algorithm compute a squarefree basis. The step limit protects the algorithm from running forever on a specialized polynomial which has a multiple root. If the step limit is too low, a squarefree basis computation will be performed unnecessarily on polynomials whose roots are simple, but close together. If the step limit is too high, time will be wasted by trying isolate a multiple root of a polynomial. In the experiments for this paper a step limit of 20 was used, and not a single real root isolation had to be preempted. A comparison of columns 2 and 3 in Table 3 shows the extent of the improvement. Listed are computing times in milliseconds for integral input polynomials of various degrees.

# 7 Reduction of Degrees

A large class of applications is concerned with complex root isolation for real polynomials $A(z)$. Thus it is useful to improve the algorithm for this special class of inputs. If $A$ is real, it suffices to isolate only roots on the real axis and in the upper half-plane. A less obvious improvement will be derived from the first two parts of the following observation:

Table 3: Improved Root Isolation

| 10-bit coefficients | | | |
|---|---|---|---|
| degree | unimproved | no sf-basis | degree reduction |
| 10 | 1883 | 1172 | 1061 |
| 20 | 13538 | 9561 | 7639 |
| 30 | 44476 | 29110 | 22882 |
| 40 | 109013 | 68285 | 56050 |
| 50 | 211539 | 141658 | 112076 |

**Lemma 1** *When $(x+iy)^n$ is distributed out,*

- *All exponents of $y$ in $\mathrm{Re}((x+iy)^n)$ are even.*
- *All exponents of $y$ in $\mathrm{Im}((x+iy)^n)$ are odd.*
- *The parities of the exponents of $x$ in $\mathrm{Re}((x+iy)^n)$ are all equal to the parity of $n$.*
- *The parities of the exponents of $x$ in $\mathrm{Im}((x+iy)^n)$ are all different from the parity of $n$.*

**Proposition 1** *Let $A(z)$ be a univariate real polynomial, let $A_1$ and $A_2$ be bivariate real polynomials with $A(x+iy) = A_1(x,y) + iA_2(x,y)$. Then*

- *All exponents of $y$ in $A_1(x,y)$ are even.*
- *All exponents of $y$ in $A_2(x,y)$ are odd.*

Consequently, for the vertically specialized polynomials the following hold:

- All exponents of $y$ in $\tilde{A}_1(y) = A_1(s,y)$ are even.
- All exponents of $y$ in $\tilde{A}_2(y) = A_2(s,y)$ are odd.

This fact can be used when critical points on a vertical edge are localized: The method described so far would isolate the real roots of $\tilde{A}_1(y)$ and of $\tilde{A}_2(y)$ in an open interval $I = (a,b)$. If the input polynomial $A(z)$ is real, all rectangles will lie in the upper half-plane, so $a \geq 0$. Hence $y \mapsto y^2$ is a one-to-one mapping from $I$ to $I^2 = (a^2, b^2)$. It induces a one-to-one mapping from the zeros of $\tilde{A}_1$ and $\tilde{A}_2$ in $I$ to the zeros of $B_1$ and $B_2$ in $I^2$, where $B_1(y^2) = \tilde{A}_1(y)$ and $B_2(y^2) = \tilde{A}_2(y)/y$. Moreover, the mapping preserves the relative locations of the zeros, and also their multiplicities: If $\eta \in \mathbf{R} - \{0\}$ is a root of $\tilde{A}_k$ of multiplicity $m$, then $\eta^2$ is a root of $B_k$ of the same multiplicity $m$, $k = 1, 2$.

Thus, the critical points, their relative locations and their classification can be obtained by isolating the real

Table 4: Times by Coefficient Length

| degree 10 | |
|---|---|
| coeff. bit-length | computing time |
| 1000 | 10133 |
| 2000 | 18294 |
| 3000 | 27368 |
| 4000 | 36431 |
| 5000 | 42862 |

Table 5: Root Approximation by Bisection

| degree 10, coeff. bit-length 10 | |
|---|---|
| accuracy | computing time |
| $10^{-10}$ | 8966 |
| $10^{-20}$ | 33733 |
| $10^{-30}$ | 56165 |
| $10^{-40}$ | 132166 |
| $10^{-50}$ | 222383 |

roots of $B_1$ and $B_2$ in $I^2$. This is an advantage since the degrees of $B_1$ and $B_2$ are half as large as the degrees of $\tilde{A}_1$ and $\tilde{A}_2$. The computing times (in ms.) in columns 3 and 4 of Table 3 document the extent of this additional improvement. A similar improvement for horizontal lines is not possible because of the last two parts of the lemma.

# 8 Average Case Complexity

Several experiments were conducted with integral polynomials with randomly generated coefficients: Table 4 strongly suggests that the average computing time (in ms.) is linearly dependent on the coefficient size of the input polynomial.

The last column of Table 3 also strongly suggests that the dependency of the average computing time (shown in ms.) on the degree of the input polynomial is cubic.

The bisection scheme algorithm can also be used to calculate the roots of the input polynomial to a given accuracy. Table 5 lists average computing times in milliseconds for various degrees of accuracy. Arguments can be given in favor of a cubic dependency of the computing time on the number of accurate digits and Table 5 is consistent with this hypothesis.

# References


[1] G. E. Collins. Infallible calculation of polynomial zeros to specified precision. *Mathematical Software, Academic Press, New York*, pages 35–68, 1977.

[2] 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.

[3] D. H. Lehmer. A machine method for solving polynomial equations. *J.A.C.M.*, 8(2):151–162, April 1961.

[4] Morris Marden. *The geometry of the zeros of a polynomial in a complex variable.* American Mathematical Society, New York, 1949.

[5] J. R. Pinkert. *Algebraic algorithms for computing the complex zeros of Gaussian polynomials.* PhD thesis, University of Wisconsin, 1973.

[6] J. R. Pinkert. An exact method for finding the complex roots of a complex polynomial. Technical Report CS-75-4, University of Tennessee, Computer Science Department, 1975.

[7] James R. Pinkert. An exact method for finding the roots of a complex polynomial. *ACM Transactions on Mathematical Software*, 2(4):351–363, December 1976.

[8] Herbert S. Wilf. A global bisection algorithm for computing the zeros of polynomials in the complex plane. *J.A.C.M.*, 25(3):415–420, July 1978.