# A tangent-secant method for polynomial complex root calculation*

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

## Abstract

Let a rectangle be given which contains exactly one complex root of a given univariate polynomial. Let the edges of the rectangle be parallel to the coordinate axes. We present a hybrid symbolic-numeric method for constructing a small subrectangle which contains the root. The subrectangle is chosen such that it contains the intersection of two triangles. The triangles are constructed so as to contain segments of certain algebraic curves that pass through the given rectangle; the intersection of these segments is precisely the complex root.

## 1 Introduction

Any univariate polynomial $A \in \mathbf{C}[z]$ can be represented canonically as a pair of bivariate real polynomials:

$$A(z) \longmapsto A(x+iy) = A_U(x,y) + iA_V(x,y)$$

The complex roots of $A$ are precisely the intersection points of the plane algebraic curves $U$ and $V$ defined by the two bivariate polynomials; $U = \{(x,y) \in \mathbf{R}^2 | A_U(x,y) = 0\}$, $V = \{(x,y) \in \mathbf{R}^2 | A_V(x,y) = 0\}$. Let $R$ be a rectangle in the complex plane with edges parallel to the coordinate axes. The intersection $I$ of the curves $U, V$ with the boundary $\partial R$ of $R$ can be computed by isolating real roots of univariate real polynomials. In [1] the authors presented a symbolic method for computing from $I$ the number of roots of $A$ in $R$, namely, by computing the winding number of $A(\partial R)$. By combining winding number computations with rectangle bisections all the roots of $A$ can be computed to specified precision.

In this paper we present a more efficient method for refining a rectangle containing exactly one polynomial root. The method is *applicable* provided each of the curves $U$ and $V$ intersect $\partial R$ in exactly two distinct regular points $u_1$ and $u_2$, and $v_1$ and $v_2$, respectively.

First, triangles $T_U$ and $T_V$ are constructed as follows: For $T_U$ compute the tangents to $U$ in $u_1$ and $u_2$; if the tangents intersect in a point $u_3$, let $T_U$ be the triangle with corner points $u_1$, $u_2$ and $u_3$, $T_V$ is constructed analogously starting from $v_1$ and $v_2$ and using tangents to $V$.

Then the triangles $T_U$ and $T_V$ are intersected with each other and with the rectangle $R$. A subrectangle $R' \subset R$ is constructed which contains $T_U \cap T_V \cap R$. The edges of $R'$ are parallel to the coordinate axes and the length of its rational coordinates is proportional to the logarithm of the lengths of its edges.

Finally, it is necessary to check whether $R'$ does contain the polynomial root. Indeed, the curve $U$ might have an inflection point in $R$, and thus the triangle $T_U$ might not contain $R \cap U$; likewise for $V$ and $T_V$.

The algorithm consists of a symbolic part, a numerical part and a hybrid symbolic-numeric subalgorithm. The winding number computations are done symbolically, because the correctness of the method depends on them. The triangles are constructed using multiprecision floating point arithmetic. Constructing them exactly would not make a qualitative difference — even an exact triangle construction might fail when $R$ contains an inflection point of $U$ or $V$ — but it would require much more computing time. The starting points $u_1, u_2, v_1$ and $v_2$ for the triangle construction are computed by a hybrid method [2], due to the authors, for calculating polynomial real roots to high precision.

*The subject matter of this paper is within the scope of POSSO

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

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

Table 1: Applicability of the Tangent-Secant Method

| rectangle size | applicability | | | | |
|---|---|---|---|---|---|
| | deg 10 | deg 20 | deg 30 | deg 40 | deg 50 |
| $2^{-1}$ | 81 % | 43 % | 21 % | 13 % | 10 % |
| $2^{-2}$ | 97 % | 83 % | 56 % | 33 % | 29 % |
| $2^{-3}$ | 100 % | 100 % | 97 % | 81 % | 72 % |
| $2^{-4}$ | 100 % | 98 % | 98 % | 94 % | 97 % |
| $2^{-5}$ | 100 % | 100 % | 100 % | 98 % | 100 % |
| $2^{-6}$ | 100 % | 100 % | 100 % | 100 % | 100 % |

Table 2: Accuracy of Real Root and Success of Tangent-Secant Step

| factor | deg 10 | deg 20 | deg 30 |
|---|---|---|---|
| 1 | 50% | 61% | 71% |
| $2^{-1}$ | 74% | 80% | 83% |
| $2^{-2}$ | 82% | 90% | 90% |
| $2^{-3}$ | 93% | 94% | 96% |
| $2^{-4}$ | 96% | 97% | 97% |
| $2^{-10}$ | 76 = 100% | 173 =100% | 268 = 100% |

# 2 Triangle Construction

Once an isolating rectangle is small enough the tangent-secant method will become applicable. In an experiment we generated some random polynomials and computed squares which contain exactly one root each. For various sidelenths we determined how many squares of that sidelength permit an application of the tangent-secant method. The percentages listed in Table 1 indicate that the tangent-secant method becomes applicable early in the refinement process.

An efficient implementation of the tangent-secant method must not use exact arithmetic. This section describes the approximate arithmetic which is used and discusses the required accuracy.

One of the coordinates of each of the points $u_1$,$u_2$,$v_1$ and $v_2$ on the boundary of $R$ is a real root of one of the univariate *specialized* polynomials obtained from the bivariate polynomials $A_U(x,y)$, $A_V(x,y)$ by specializing one of the variables to the coordinates of $R$. For example, let $E = \{(s,y) \mid y \in (t_1,t_2]\}$, $s,t_1,t_2 \in \mathbf{Q}$ be one of the vertical edges of $R$. Then the intersection points $u_1$ and $u_2$ of $E$ with the curve $U$ are of the form $u_1 = (s,\alpha)$, $u_2 = (s,\beta)$, where $\alpha$ and $\beta$ are the real roots of the univariate polynomial $A_U(s,y)$ in the interval $(t_1,t_2]$. While the algorithm is set up such that $s$, $t_1$ and $t_2$ will always be *binary rational numbers* (i.e. numbers of the form $a2^b$, with $a,b \in \mathbf{Z}$), $\alpha$ and $\beta$ will, in general, be irrational algebraic numbers.

An exact implementation of the tangent-secant method would have to compute the slope $(\triangle x, \triangle y)$ of the tangent to $U$ in $u_1$ by evaluating

$$(\triangle x, \triangle y) = (\frac{\partial A_U}{\partial x} u_1, \frac{\partial A_U}{\partial y} u_1); \tag{1}$$

it would then intersect the triangles by computing in $\mathbf{Q}(\alpha,\beta,\gamma,\delta)$, where $\alpha$,$\beta$,$\gamma$ and $\delta$ are the algebraic number coordinates of the points $u_1$,$u_2$,$v_1$ and $v_2$.

Our experiments show that in practice the tangent-secant method will work with far less expensive, approximate calculations. The algebraic number coordinates can be computed to any desired accuracy using floating point arithmetic of appropriate precision. Likewise, the tangent slopes can be calculated using floating point arithmetic.

# 3 Real Root Refinement

The tangent-secant method reduces complex root refinement to real root refinement. In each complex root refinement step the real algebraic number coordinates of the points $u_1$,$u_2$,$v_1$ and $v_2$ are approximated to about the same accuracy which is expected from the tangent-secant step, namely, $K\epsilon^2$, where $K$ is an unknown constant and $\epsilon$ is the sidelength of the current rectangle $R$.

Table 2 shows the success rate of the tangent-secant construction for various values of $K$: Twenty polynomials with 10-bit integer coefficients were randomly generated for each of the degrees 10, 20 and 30. Their roots in the upper half-plane were isolated in squares of sidelength $2^{-10}$. (The 20 polynomials of degree 30 had a total of 268 roots in the upper half plane). For each isolating square the tangent-secant method was applicable, and one tangent-secant step was carried out. The tangent slopes were calculated with high precision, but the points $u_1$,$u_2$,$v_1$ and $v_2$ were approximated only with accuracy $K2^{-20}$ for $K = 2^{-j}$ with $j = 0,1,2,3,4,10$. Whenever the tangent-secant step failed this was due to insufficient approximation of real roots; in those cases a subrectangle was produced with sides shorter than $2^{-11}$ which did not contain the complex root. The entries in the table are the percentages of rectangles for which the tangent-secant step was successful.

The algorithm used for real root calculation is a small-scale model of the proposed method for complex root calculation: Both start with an exact method and continue with numerical steps with *a posteriori* verification of the results. In case a result is incorrect the algorithm falls back to an exact method. A detailed description of the method for real root isolation can be found in [2].

# 4 Slope Computation

The tangent slopes are computed according to Equation (1) using floating point arithmetic.

The precision for this computation is set according to the accuracy $e$ which is expected from the entire tangent-secant construction. As in Section 3, $e = 2^{-j}\epsilon^2$, where $j$ is a small positive constant and $\epsilon$ is the sidelength of the current rectangle $R$. The mantissa length for the floating point calculations is set to $n$ units of $\zeta$ bits each, where $\zeta$ depends on the wordlength of the computer. In our implementation, $\zeta = 29$. If $e$ is the expected accuracy we choose $j = 10$ and

$$n = \left\lceil \frac{\log_2 \frac{2^j}{\epsilon^2}}{\zeta} \right\rceil . \tag{2}$$

# 5 Intersecting the Triangles

Given a rectangle $R$ whose edges are parallel to the coordinate axes and two triangles $T_U$, $T_V$ such that two vertices of each triangle are on the boundary of $R$ the vertices of the intersection polygon have to be computed.

Each of the vertices of the intersection polygon is either a vertex of one of the input polygons or an intersection point of two edges from different input polygons. So, it suffices to check these points for containment in the other input polygons. Of these points the ones contained in all of the input polygons form a convex span for the intersection polygon.

Computing the intersection point of two edges essentially involves computing $(2\times2)$-determinants. Containment of a point $P$ in a triangle is decided by evaluating signs of line equations $L$ at points $P$. The sign of $L(P)$ is called the *sense* of $P$ with respect to the line $L$. If — for all three edges of the triangle — the sense of $P$ with respect to the edge agrees with the sense of the triangle vertex which is not on the edge, $P$ is contained in the triangle.

In our experiments these computations were carried out exactly using binary rational numbers.

# 6 Subrectangle Construction

Given is a set $S$ of (intersection) points inside an axes-parallel rectangle $R = [X_1, X_2] \times [Y_1, Y_2]$ with binary rational coordinates $X_1 < X_2$ and $Y_1 < Y_2$. A subrectangle $R'$ of $R$ has to be found such that $R'$ is a non-degenerate rectangle which contains $S$. The rectangle $R'$ should be as small as possible, and its coordinates should be short binary rational numbers.

The *bounding box* $B = [x_1, x_2] \times [y_1, y_2]$ of $S$, i.e. the smallest axes-parallel rectangle containing $S$, is obtained by determining the minimum and the maximum of the $x$- and $y$-coordinates of the points in $S$. If $x_1 = x_2$, B is replaced by

$$[\max(X_1, x_1 - \epsilon^2), \min(X_2, x_2 + \epsilon^2)] \times [y_1, y_2],$$

where $\epsilon$ is the length of the longer side of $R$. Analogously for $y_1 = y_2$. In this fashion, a non-degenerate subrectangle $B$ of $R$ is obtained which contains the points of $S$.

In a final step $B$ is enlarged slightly to form the rectangle $R'$ with binary rational coordinates of lengths commensurate with the sidelengths of $R'$. This step is guaranteed to keep $R'$ contained in $R$. Let $I = [a, b]$ be the projection of $B$ to the $x$-axis. Let

$$c = \left\lfloor \frac{a}{2^{\lfloor \log(b-a) \rfloor -1}} \right\rfloor 2^{\lfloor \log(b-a) \rfloor -1} \tag{3}$$

and

$$d = \left\lceil \frac{b}{2^{\lfloor \log(b-a) \rfloor -1}} \right\rceil 2^{\lfloor \log(b-a) \rfloor -1}. \tag{4}$$

Then $J_x = [c, d]$ is an interval with binary rational endpoints containing $I$ and contained in $[X_1, X_2]$. The projection of $B$ to the $y$-axis is processed analogously, rendering an interval $J_y$. Now $R' = J_x \times J_y$ is a subrectangle of $R$ with the desired properties.

**R 1** 1.0, 1.0, 1.25, .75, 1.0, 1.33, .75.

**R 2** 1.0, 1.5, 2.0, 1.25, 2.0, 1.25.

**R 3** 1.0, 1.2, .6, .83, .75, .75.

**R 4** 1.0, .046875, 51.2, .015625, 64.0, .015625, 51.2.

**R 5** 1.0, 1.0, .75, 1.0, 1.0, .8.

**R 6** 1.0, 2.0, 1.2, 1.25, 1.5, 1.5, 1.6.

**R 7** 1.0, .75, 1.0, .83, 1.0, .8.

**R 8** 1.0, .375, .75, .4166, .66, .375.

**R 9** 1.0, 1.5, 1.0, 1.33, 1.0, 1.0.

**R 10** 1.0, 4.0, .125, 8.0, .15, 6.4, .125.

**R 11** 1.0, .4, 1.0, .6, 1.33, .5.

**R 12** 1.0, 1.25, 1.25, 1.25, 2.0, 1.33, 1.2.

**R 13** 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0.

**R 14** 1.0, 1.5, 1.66, 1.33, 1.0, 2.0.

Figure 1: Sidelength Ratios During Refinement

Table 3: Rate of Convergence

| From 10 to 300 Bits of Accuracy | |
|---|---|
| degree | tangent-secant steps per root |
| 10 | 5 |
| 20 | 5.6 |
| 30 | 5.4 |

Table 4: Tangent-Secant Method vs. Bisection

| refinement | | deg 10 | | deg 20 | | deg 30 | |
|---|---|---|---|---|---|---|---|
| from | to | TS | B | TS | B | TS | B |
| $2^{-10}$ | $2^{-20}$ | 3.8 | 2.6 | 18.8 | 20.9 | 74.8 | 81.4 |
| $2^{-20}$ | $2^{-40}$ | 3.8 | 10.3 | 36.3 | 85.9 | 181.2 | 363.9 |
| $2^{-40}$ | $2^{-80}$ | 9.1 | 41.8 | 118.2 | 349.7 | 478.1 | 1,602.2 |
| $2^{-80}$ | $2^{-160}$ | 26.1 | 199.1 | 349.6 | 1,728.7 | 1,274.3 | — |

Table 5: Breakdown of Refinement Time

| degree 10, 10-bit coefficients<br>refinement from 10 to 300 bits of accuracy | |
|---|---|
| computation | % of total time |
| real root refinement | 41% |
| slope computation | 2% |
| intersection of triangles | 4% |
| winding number computation | 50% |
| others | 3% |

Figure 1 reports anecdotical evidence about the shape of the rectangles obtained during the refinement process. Listed are the ratios

$$\frac{\text{vertical sidelength}}{\text{horizontal sidelength}}$$

for the rectangles obtained by successive refinements from 14 isolating rectangles. These 14 isolating rectangles are squares of sidelength $2^{-10}$. Together they isolate all roots in the upper half plane of a certain polynomial of degree 30 with randomly generated 10-bit coefficients. The rectangles are refined until both of their edges are shorter than $2^{-300}$. Eight of the rectangles need five refinement steps each, six rectangles need six refinement steps. Most subrectangles are about square. All refinements of rectangle 13 are squares. The sidelength ratio of the refinements of rectangle 4 oscillates wildly, perhaps because the (orthogonal) curves $U$ and $V$ happen to be locally about parallel to the coordinate axes: If in this situation $R$ happens to be much wider than high, $R'$ will be much higher than wide, etc.

# 7 Performance

Table 3 lists — for various polynomial degrees — the number of tangent-secant steps which were necessary to refine an isolating rectangle of sidelength $2^{-10}$ to sidelength $2^{-300}$. For a quadratically convergent method $\lceil \log_2 \frac{300}{10} \rceil = 5$ refinement steps would be expected, and the tangent-secant method agrees well with this expectation.

Table 4 shows that already at relatively small refinement tasks the tangent-secant method becomes superior to straightforward bisection. The table contrasts computing times for the tangent-secant method with those for the bisection method. All input polynomials have randomly generated 10-bit integer coefficients. All times are shown in seconds. In the computations carried out for the table the tangent-secant method never failed: It was applicable to all rectangles $R$ for which it was invoked; in each case it produced a subrectangle $R'$ which contained the complex root. The timings were obtained on a DECstation 5000/200 running ULTRIX V4.0. The algorithms were implemented in Portable C on top of the `SACLIB`-library (version 1.1) of computer algebra programs.

A breakdown of the computing time for the tangent-secant method is given in Table 5. The table lists the percentage of the total complex root refinement time spent on the various parts of the computation; the figures were obtained when rectangles of sidelength $2^{-10}$ isolating roots of a polynomial of degree 30 with 10-bit integer coefficients were refined to sidelength $2^{-300}$. The winding number computation is the costliest part of the algorithm since it uses a symbolic method. The numerical slope calculations scarcely contribute to the total time; neither do the binary rational calculations that were used for triangle intersection. The hybrid subalgorithm for real root refinement that we actually used for this table uses exact calculation for *a posteriori* verification instead of floating point interval arithmetic; thus, the time for that part of the computation can be significantly improved.

# References


[1] George E. Collins and Werner Krandick. An efficient algorithm for infallible complex root isolation. In Paul S. Wang, editor, *Proceedings of the 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 and Werner Krandick. A hybrid method for high precision calculation of polynomial real roots. In Manuel Bronstein, editor, *Pro-*

*ceedings of the International Symposium on Symbolic and Algebraic Computation*, pages 47–52, ACM Order Department, P.O.Box 64145, Baltimore, MD 21264, Phone: 1-800 342-6626 or 1-410 528-426, 1993. ACM.