Empirical Study of an Evaluation-Based Subdivision Algorithm for Complex Root Isolation

Narayan Kamath
Google, Inc
and
Oxford Computing Lab, UK
kamath@kellog.ox.ac.uk

Irina Voiculescu
Oxford Computing Lab
Oxford University, UK
irina@comlab.ox.ac.uk

Chee K. Yap*
Courant Institute, NYU, USA
and
Oxford Computing Lab, UK
yap@cs.nyu.edu

ABSTRACT

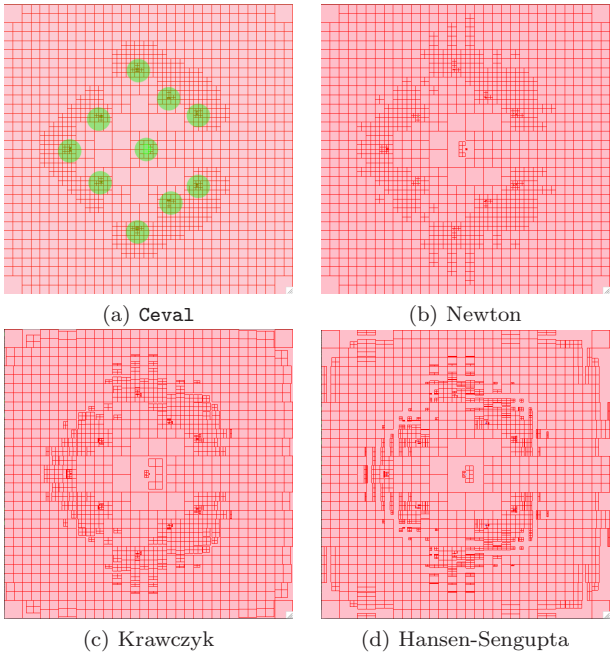
We provide an empirical study of subdivision algorithms for isolating the simple roots of a polynomial in any desired box region B_0 of the complex plane. One such class of algorithms is based on Newton-like interval methods (Moore, Krawczyk, Hansen-Sengupta). Another class of subdivision algorithms is based on function evaluation. Here, Yakoubsohn discussed a method that is purely based on an exclusion predicate. Recently, Sagraloff and Yap introduced another algorithm of this type, called `Ceval`. We describe the first implementation of `Ceval` in Core Library. We compare its performance to the above mentioned algorithms, and also to the well-known `MPSolve` software from Bini and Florentino. Our results suggest that certified evaluation-based methods such as `Ceval` are encouraging and deserve further exploration.

1. INTRODUCTION

Isolating roots of univariate polynomials is a highly classical subject in the mathematical and computational fields. From a complexity viewpoint, the problem of isolating all complex zeros of an integer polynomial (we call this the "benchmark problem") has been intensely studied [30, 31, 25]. The basic conclusion from Schönhage's classic paper [30] says the benchmark problem has bit-complexity $\widetilde{O}(n^3L)$ where n is the degree and L a bound on the bit sizes of each coefficient ($\widetilde{O}$ means we ignore logarithmic terms in n and L). However, practitioners tend to favor other methods that do not match this asymptotic complexity. Among the many current implementations, we have the highly regarded `MPSolve` from Bini and Florentino [2], and an optimized Descartes method from Rouillier and Zimmermann [28], albeit for real roots. For instance, `Maple`'s default algorithm for finding real roots is based on the latter implementation [28]. Complex roots of univariate polynomials can be viewed as a special case of solving bivariate real systems. The latter has been the focus of several recent papers [1, 11, 8, 7]. Our work can also be regarded as a special case of the general problem of determining the topology of a collection of planar curves.

This paper is an empirical study of several complex root isolation methods based on **domain subdivision**. The basic paradigm is repeated subdivision of an initial box $B_0 \subseteq \mathbb{R}^2$, analogous to binary search. In Figure 1(a,b,c,d), we provide a visualization of the subdivision boxes produced by four of the algorithms to be discussed. The polynomial whose roots were isolated here is $x(9x^9 + 7x^8 + 8x^7 + 8x^6 + 6x^4 + 5x^3 + 2x^2 + 1) = 0$.

Many algorithms in this area can be viewed as having 2



(a) `Ceval` (b) Newton (c) Krawczyk (d) Hansen-Sengupta

Figure 1: Roots of $x(9x^9 + 7x^8 + 8x^7 + 8x^6 + 6x^4 + 5x^3 + 2x^2 + 1) = 0$

*This author's work is performed on a sabbatical visit to Oxford Computing Lab (2009-10) and is supported by NSF Grants CCF-0728977 and CCF-0917093.

TO APPEAR: SNC 2011, June 7-9, 2011, San Jose, California.

or 3 phases, beginning with a subdivision phase. See [16] for a description of this framework. Such algorithms goes back to an algorithm of Mitchell [17] for finding real roots of univariate polynomials to higher dimensional analogues for computing isotopic curves and surface approximations [26, 32]. This phase is controlled by an associated predicate $C(B)$ where $B \subseteq B_0$ is any axes-parallel box. Starting with B_0, we keep subdividing a box $B \subseteq B_0$ into two or four subboxes until every box B satisfies $C(B)$. The complexity of this subdivision phase often determines the asymptotic complexity of the algorithm. Subdivision algorithms are popular with implementers because of merits such as ease of implementation, local complexity, and adaptive complexity. By "local complexity", we mean that the computational effort can be localized to a region-of-interest like B_0; in contrast, the benchmark problem has global complexity. We remark that the predicate $C(B)$ can be algebraic or geometric ones (cf. [16]) but we are most interested in numeric ones.

The subdivision predicate $C(B)$ is typically a disjunction of an **exclusion** $C_{out}(B)$, and an **inclusion** $C_{in}(B)$, predicate. If a box is excluded, i.e., satisfies $C_{out}(B)$, it may be discarded, unless we wish to hold it for visualization purposes. Ultimately, we need a **confirmation predicate** which determines that box B has a unique root. We remark that $C_{in}(B)$ is a necessary but not sufficient condition for confirmation (this is dealt with in subsequent phases). Conceptually, the set of boxes form a **subdivision tree** T, and the the goal of subdivision is to ensure that every leaf of T is either excluded or included. The **effectiveness** of a predicate $C(B)$ is measured by the overall complexity of the subdivision process. Effectiveness of numerical predicates is seen to depend on two factors: **(predicate) efficiency** and **efficacy**. Efficiency measures the computational effort to evaluate each predicate. Efficacy measures the size of the subdivision tree T. The effectiveness of a predicate can be measured (lower bounded) by the product of the size of T and worst case cost to evaluate any predicate in T. However, it is shown in [29] that such a product bound may be a factor of n larger than the true overall complexity of subdivision. In general, effectiveness involves a tradeoff between efficacy and (predicate) efficiency. Such a efficiency-efficacy tradeoff for real root isolation is discussed in [29] (in the context of comparing predicates based on Sturm, Descartes and Bolzano principles). Numerical predicates are typically constructed from interval functions, and so predicate efficacy depends on the range of interval functions; Stahl [33] contains a comprehensive study of range functions from this efficacy-complexity viewpoint, and also its application to the solution of systems of nonlinear equations. This paper is a contribution along similar lines, except that we deal with the specific case of a bivariate system from the real and imaginary curves $u(x, y)$, $v(x, y)$ of a complex polynomial $p(x + \mathbf{i}y) = u(x, y) + \mathbf{i}v(x, y)$ ($\mathbf{i} = \sqrt{-1}$).

We first look at a class of predicates from the certified computation literature [19, 23]. For any real function $f : \mathbb{R}^n \to \mathbb{R}^m$, we postulate a corresponding interval function $\square f : \square\mathbb{R}^n \to \square\mathbb{R}^m$ where $\square\mathbb{R}$ is the set of closed real intervals. Typically, $m = 1$ or $m = n$. We call $\square f$ a **box function** for f if it satisfies two properties: for all n-boxes $B, B_i \in \square\mathbb{R}^n$, (a) [Inclusion Property] $f(B) \subseteq \square f(B)$ and, (b) [Convergence Property] for any point $p \in \mathbb{R}^n$, we have $\square f(B_i) \to f(p)$ as $B_i \to p$ ($i \to \infty$). Although Newton's method is traditionally used for root refinement, Nickel [24], Moore [20], and Hansen [13] have shown that the interval forms can serve as confirmation predicate for roots. Such predicates form the basis for root isolation algorithms [18, 33]. We focus on three forms of Newton-type predicates: an interval Newton predicate due to Moore [19], Krawczyk's predicate [20, 15], and Hansen and Sengupta's predicate [12]. These Newton operators have the form $f : \mathbb{R}^n \to \mathbb{R}^n$ and their inclusion/exclusion predicates have the form

$$\left.\begin{array}{ll} \text{(inclusion)} & \square f(B) \subseteq B \\ \text{(exclusion)} & \square f(B) \cap B = \emptyset \end{array}\right\} \tag{1}$$

where B is a box and $\square f$ is a box function for f. Thus, the efficacy of these predicates may be reduced to the following order relation $\succeq$ on their underlying operators: we write "$\square f \succeq \square g$" if $\square f(B) \subseteq \square g(B)$ for all boxes B. Thus, $\square f \succeq \square g$ implies that, for predicates of the form (1), the predicates defined by $\square f$ will succeed whenever the predicates defined by $\square g$ succeed. I.e., $\square f$ is at least as efficacious as $\square g$. From Neumaier [23], we have:

$$\text{Interval Newton} \succeq \text{Hansen-Sengupta} \succeq \text{Krawczyk}.$$

The corresponding Krawczyk predicate is therefore the least efficacious among the three. However, these operators do not have equal computational costs: the interval Newton operator involves the inversion of an interval matrix, while the other two operators do not. Our aim is to implement all three predicates on a common platform, and compare their overall performance in isolating the roots of a wide variety of polynomials.

Another source of predicates comes from function evaluation [10, 36, 34, 6, 5]. Yakoubsohn and Dedieu [10] studied an exclusion predicate $C_{out}(B)$ that is basically an interval form of Taylor expansion. Their inclusion predicate $C_{in}(B)$ amounts to the ε-cutoff predicate (i.e., B is included if its diameter is $< \varepsilon$). Since ε is arbitrary, some adaptivity is lost and the algorithm only produces a collection of candidate boxes, which may contain no roots or multiple roots. Recently, Sagraloff and Yap [29] introduced another evaluation-based algorithm `Ceval`, but with inclusion and confirmation predicates. We provide the first implementation of `Ceval`, and compare these evaluation-based algorithms to each other, and to the interval Newton-type predicates above. We note that for the benchmark problem, the paper [29] proved that `Ceval` has bit complexity of $\widetilde{O}(n^4L^2)$, thus matching the best bit complexity for Descartes method or Sturm methods. However, the `Ceval` complexity is a stronger result, since Descartes and Sturm methods only isolate real roots.

The last algorithm in our comparative study is `MPSolve`, a complex root algorithm from Bini and Florentino [2, 4]. This is a highly regarded implementation based on the Aberth-Erlich method, not subdivision. As such it is a global method (it must approximate all roots simultaneously). But no convergence proof is known for this method.

Our implementation is released as part of the free and open source `Core Library` [38, 9]. We exploit the unique ability of `Core Library` to compile a program in different levels of numerical accuracy. The work reported here is based on the first author's Masters Thesis [14].

Postscript. A "simplified" version of `Ceval` was proposed in the latest version of the paper [29]. The experiments reported here are based on the original version of `Ceval`. In the full version of this paper, we plan to report on the relative performance of these two versions.

2. CONTRIBUTIONS

The contributions of this paper are:

- An implementation of three interval Newton-type root isolation algorithms for bivariate systems on a common platform. We report on their relative performance for complex root isolation.
- The first implementation of the `Ceval` algorithm.
- Comparison of evaluation-based subdivision methods (Yakoubson, `Ceval`) with Newton-type methods and with a state-of-art root solver, `MPSolve`. We discuss algorithm engineering issues arising in these implementations.
- Our empirical evidence suggests that evaluation-based methods for complex roots are vastly superior to the (more general) Newton-type methods, and has competitive advantages compared with current state-of-art methods such as `MPSolve`.
- Our present work is a contribution to the general area of designing new exact and certified algorithms for approximation of zero sets using evaluation-based methods.
- Distribution of our code with the free open-source `Core Library`, allowing further experimentation.

3. EXPERIMENTAL SETUP

The implementation platform is a MacBook Pro with an Intel Core 2 Duo processor, clocked at 2.53 GHz with 4 GB of RAM. The operating system is `Mac OS X` (10.5.8). Our compiler is `gcc-4.2`, and our code is compiled using its most aggressive optimization flag `-O3`.

Our algorithms are implemented as `C++` programs using the `Core Library` version 2.0. Recall that the `Core Library` [38] is a collection of `C++` classes to support exact geometric computation (EGC) [37] with algebraic numbers. It is built over the multiprecision libraries of `gmp` and `mpfr`. A unique aspect of `Core Library` is its **numerical accuracy API**, which comes in four levels. Level 1 represents machine accuracy (IEEE 754 Standard). Level 2 represents arbitrary precision number types such as `BigInt` and `BigFloat`. Level 3 represents the "EGC Level" and the main number type here is called `Expr` (Expression). Exact comparison of an `Expr` is achieved as long as `Expr` is algebraic. Level 4 allows a mix of number types from all 3 previous levels. These four levels are integrated in the sense that any program that includes `Core Library` is able to compile its numbers to any these four levels. For instance, a number declared as "`double`" represents the standard machine double-precision number at Level 1, but it is promoted to a `BigFloat` in Level 2, and promoted to an `Expr` in Level 3. This exploits the operator overloading feature of the `C++` language. We mainly use Levels 1 and 2. Level 1 is fast, but it limits the size of the polynomials that can be treated. All our experiments are assumed to run at Level 1 unless otherwise indicated.

For this work, two new number types `IntervalT` (intervals) and `ComplexT` (complex numbers) are essential. `Core Library`'s polynomial classes are generic (i.e., templated) and this allows us to evaluate a polynomial $p(x)$ at interval valued x or complex number x, independent of the number type of p's coefficients:

```
template <typename T> class Polynomial {
  // Note that T and NT must be
  // either interoperable
  // or implicitly convertible.
  template <typename NT>
       NT eval(const NT &x);
};
```

The compiler can instantiate for `NT=IntervalT`, `NT=ComplexT` or `NT=BigFloat`. Typically, we have `T=BigInt` or `T=int`.

Our algorithms are exact because all predicates can be implemented exactly using `BigFloats`, assuming the input numbers are dyadic (i.e., numbers of the form $m2^n$ where $m, n \in \mathbb{Z}$). Level 1 accuracy is sufficient for exact computation provided no overflow or underflow occurs. In the implementation of `Ceval`, the 8-point test requires rational numbers, but this will be separately treated below. Speed without correctness is not worth much. So we validate the output of our algorithms in two ways: we compare our computed values to a reliable software such as `MPSolve`, and we compare Level 1 outputs to the outputs from the same algorithm at Level 2.

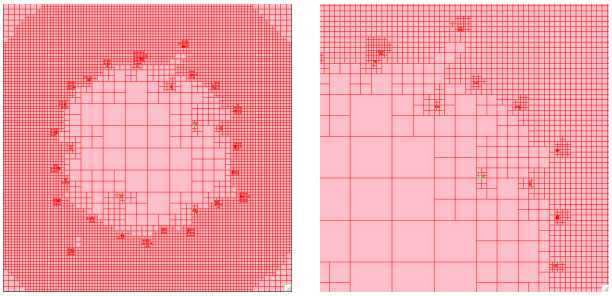
The main sources for our test polynomials are the `FRISCO` suite [35], and randomly generated polynomials. Our programs accepts both the `FRISCO` file format for polynomials as well as a very flexible string based format such as "$3x^4y^5 - (x^2+1)(y^3-1)$" or "$3x^4y^5 = (x^2+1)(y^3-1)$", which is useful for command line execution. For each run of our algorithm, we collect the following statistics:

- Running Time in microseconds, using `C`'s `gettimeofday API`. This method is sufficiently reliable for our purposes because our programs are CPU intensive and do not perform any disk or network I/O.
- Number of Boxes Processed. In tables, this number is called "Iters" (as it is the number of iterations of the subdivision loop). Recall that each box is ultimately included, excluded, or subdivided.
- Number of Excluded (resp., Included) Boxes.
- Number of Unresolved Boxes. In principle, our algorithm will eventually terminate if there are no multiple roots in the region of interest. But to prevent underflow (level 1), or to keep the running time reasonable, or to avoid nontermination in case the region-of-interest contains multiple roots, we can specify a minimum box size or maximum tree depth; this may result in unresolved boxes on termination.

In our statistical tables, we will underline the entry that is the best among the several methods being compared. Also, the names of the polynomials (for instance, `chebyshev20`) have an integer suffix indicating its degree.

4. VISUALIZATION

Visualization is an important aspect of our software, for debugging and for gaining insight into the behavior of algorithms. Here we extend the previous visualization for curves analysis from [16], which is based on the `OpenGL` library. Typically we wish to visualize a subdivision tree (Figure 2(a)) and to zoom and pan into details of interest (Figure 2(b)). To do this we need to retain boxes that have been excluded in subdivision; command line flags control the



(a) All roots (b) Zoomed in view.

Figure 2: Subdivision tree of `Ceval` for degree 25 polynomial

handling of these extra information. The color code shows red for excluded boxes, green for confirmed boxes, blue for unresolved boxes.

Although the termination of the `Ceval` algorithm depends on the assumption that the input polynomial is square-free, the algorithm is still useful for arbitrary polynomials. Suppose we run `Ceval` on the polynomial $p(z) = (z^6+64)^2(z^6-729)$. This is a non-squarefree polynomial with 6 multiple roots on the circle (centered at the origin) of radius 2, and 6 simple roots on a concentric circle of radius 3. It is now essential to specify a minimum box size (or maximum subdivision depth) for termination. Choosing a minimum box size of 0.0001, and $B_0 = [-4,4]\times[-4,4]$, `Ceval` produces the visualization seen in Figure 3. It processed a total of 8645 boxes, confirming 6 green boxes and leaving unresolved 256 blue boxes. The simple roots were all isolated as shown in this output:

```
m= [1.498046875 + (-2.599609375)i], r= .01171875
m= [-1.501953125 + (-2.599609375)i], r= .01171875
m= [2.998046875 + (.001953125)i], r= .01171875
m= [1.498046875 + (2.599609375)i], r= .01171875
m= [-3.001953125 + (.001953125)i], r= .01171875
m= [-1.501953125 + (2.599609375)i], r= .01171875
```

Further details about our implementation may be found in the thesis [14] and in the `Core Library` distribution (under `progs/mesh`).

5. INTERVAL NEWTON-TYPE METHODS

First, we consider three closely related pairs of predicates based on Interval Newton methods. They work by finding the solutions of the bivariate system $u(x,y) = v(x,y) = 0$ where $f(x+\mathbf{i}y) = u(x,y)+\mathbf{i}v(x,y)$. But these methods work more generally for any zero-dimensional system of n polynomials in n variables, so we describe them in these general terms. We view an n-tuple B of vectors as an n-dimensional box (or n-box), $B \subseteq \square\mathbb{R}^n$. The **Interval Newton operator** $N : \square\mathbb{R}^n \to \square\mathbb{R}$ for a function $f : \mathbb{R}^n \to \mathbb{R}$ is given by

$$N(B) = N_f(B) := m(B) - (\square J(B))^{-1} \cdot f(m(B)) \qquad (2)$$

where $(\square J(B))^{-1}$ denotes the inverse of the interval Jacobian of f, and $m(B)$ denotes the midpoint of box B. For bivariate systems, computing the inverse is not really an issue, but we must address the situation where $\square J(B)$ contains 0. The **Krawczyk operator** is defined by

$$K(B) = K_f(B) := y - Y\cdot f(y) + \{I - Y\cdot\square J(B)\}\cdot(B-y) \quad (3)$$

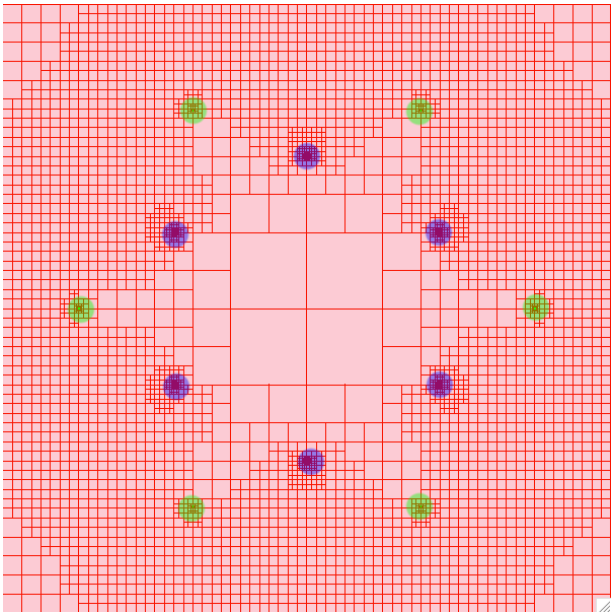


Figure 3: Subdivision tree for $p(z) = (z^6+64)^2(z^6-729)$. Blue boxes are unresolved ones around double roots, and green boxes isolates the simple roots.

where $y \in B$, Y is any nonsingular real matrix, and I the identity matrix. Typically, we have $y \simeq m(B)$ and $Y \simeq J(y)^{-1}$, viewed as a preconditioner. The Hansen-Sengupta approach is a Gauss-Seidel iteration to solve for x in the equation

$$\square J(B)(x-y) + f(y) = 0. \qquad (4)$$

Multiplying by some preconditioning matrix Y as before, the equation becomes $A(x-y) = -Yf(y)$ where $A := Y\cdot\square J(B)$ (cf. (3)). We can write A as $A = U+D+L$ (disjoint sum of a strict upper triangular U, a diagonal D, and a strict lower triangular L matrix). The Gauss-Seidel iteration for solving $Ax = b$ is given by

$$x^{(k+1)} = D^{-1}(b - Lx^{(k+1)} - Ux^{(k)})$$

where $x^{(k)}$ is the k^{th} approximation. This yields the **Hansen-Sengupta operator**

$$H(B) = H_f(B) := y - D^{-1}\{Y\cdot f(y) + L(B'-y) + U(B-y)\} \qquad (5)$$

where y, Y are as before and $B' = B\cap H(B)$. Note that the use of LB' in (5) is not recursive, but iterative because L is lower-triangular: we replace each component of box B by corresponding entries of $B\cap H(B)$ as they become available.

Let G be any of the three operators from (2,3,5) above. It can be shown [19, 24, 20, 21] that

(i) $G(B) \subseteq B$ implies B has a unique root

(ii) $G(B)\cap B$ contains all the roots in B

As a consequence of (ii), if $G(B)\cap B = \emptyset$ then B has no roots. This provides an exclusion predicate C_{out} for subdivision. Clearly (i)) provides a confirmation predicate, and hence an inclusion predicate C_{in}. Since this inclusion predicate is also a confirmation, the interval Newton-type algorithms do not need additional phases beyond the subdivision phase.

Another consequence of (ii) is that, in case the inclusion or exclusion predicates fail on B, we can replace B by $B' := B \cap G(B)$ and then subdivide B'. Although this results in smaller boxes, there is an associated cost because the bitsize (of the coordinates) of B' may increase considerable. If we simply subdivide B, the bitsize increases by a small constant number (depending on how you count bitsize of B). The effect of using B' instead of B implies that the shapes of boxes are rather arbitrary. This results in irregular children boxes as seen in Figure 1(b,c) but especially Figure 1(d) because of extended interval arithmetic.

§1. The Exclusion Predicate $C_0(B)$.

Instead of the exclusion predicate $G(B) \cap B = \emptyset$ above, we can use predicate $C_0(B)$, defined as $0 \notin \square u(B)$ or $0 \notin \square v(B)$. (cf. [16]). Preliminary experiments show that $C_0(B)$ is more efficient, and all our Newton-type algorithms use this predicate. The interval $\square u(B)$ can be computed easily using the Horner scheme. But our tests show that using centered form [27] of $\square u(B)$ yields remarkable improvement in efficacy over Horner. This becomes more apparent with increasing degree of the polynomial u. This gain in efficacy must be balanced against the cost of computing centered forms, which amounts to computing the bivariate Taylor coefficients of our polynomials when expanded at the midpoints of a box B. We achieve a balance by using a simple scheme: recompute the coefficients after a fixed number T (threshold) of predicate failures. It is worth pointing out that this strategy is sensitive to the order in which we process boxes, a "depth first" type processing will yield better results than a "breadth first" type processing. Table 1 shows the influence of T on timing and subdivision for the Chebyshev polynomial of degree 20. For this particular example, $T = 16$ gives the best overall time (indicated by the underlined entry). As a general default, we have empirically set $T = 4$ in all our experiments below.

T	Iterations	Ambiguous boxes	C0 Excludes as % of total	Time (secs)
0	15429	176	11396(73.8)	18.554
2	19845	504	13877(71.2)	3.410
4	23585	744	15218(64.5)	2.190
8	31229	1332	16936(54.2)	1.834
16	40585	2396	17201(42.3)	1.720
32	56945	5908	17700(31.0)	1.951
64	77949	9164	17772(22.7)	2.445

Table 1: Effect of Threshold T on Chebyshev Polynomial of degree 20

§2. Comparing the Inclusion Predicates.

We now return to the inclusion predicate $G(B) \subseteq B$ where $G(B)$ is one of the operators $N(B), K(B)$ or $H(B)$. These operators admit several variants in their implementation. For the interval Newton operator (2), when $J(B)$ contains a singular matrix, we have a choice to use extended interval arithmetic (to support division by intervals containing zero) or we could choose to subdivide B. The latter turns out to be a consistently better choice. For the Krawczyk operator (3), we could choose the nonsingular matrix Y to be one of the following: $J(m(B))^{-1}$, $m(\square J(B)^{-1})$, or identity I. The first choice turns out to be best. For the Hansen-Sengupta operator we choose to use extended interval arithmetic in its implementation. See [14] for more detail. In any case, we choose the best variant for each inclusion predicate and then compare them against each other. This final comparison is shown in Table 2.

Polynomial	Hansen-Sengupta		Newton		Krawczyk	
	# iter	Secs.	# iter	Secs.	# iter	Secs.
cheby20	18837	24.448	15445	19.731	15429	20.064
chrma22	7621	10.986	6557	9.380	6637	9.578
chrmc23	11229	19.637	9893	16.191	9845	15.328
hermite20	2805	3.453	2645	3.278	2645	3.344
kam3-1	2005	0.439	2005	0.438	2005	0.435
kam3-2	2005	0.444	2005	0.432	2005	0.446
laguerre20	1205	1.491	1037	1.294	1021	1.272
laguerre4	205	0.011	229	0.013	213	0.012
laguerre5	197	0.016	197	0.021	197	0.017
laguerre6	245	0.030	245	0.149	245	0.036
$x^{10}-1$	3253	1.100	2645	0.796	2629	0.809
$x^{20}-1$	12693	15.729	10053	12.373	10053	12.267
wilk20	1133	1.433	869	1.110	861	1.096

Table 2: Comparison of Hansen-Sengupta, Newton, and Krawczyk

The three operators are not very different from each other. It is somewhat surprising that Hansen-Sengupta did not perform better, but this may be because our special problem of complex roots does not allow the Hansen-Sengupta method to shine. Another issue is that these methods do not converge in case of roots on the boundary of a subdivision box; this is treated by Stahl [33] and also in Kamath's thesis [14].

6. CEVAL & YAKOUBSOHN'S ROOT ISOLATION ALGORITHM

For any function $f : \mathbb{C} \to \mathbb{C}$, and constant $K > 0$, Sagraloff and Yap [29] introduced the function $t^f_K : \mathbb{C} \times \mathbb{R}_{\geq 0} \to \mathbb{R}$ defined as follows:

$$t^f_K(m, r) = K \sum_{k \geq 1} \left| \frac{f^{(k)}(m)}{f(m)k!} \right| r^k. \tag{6}$$

The function $M(z, t)$ introduced by [36, 10] corresponds to the case $K = 1$. The predicate $T^f_K(m, r)$ is then defined as "$t^f_K(m, r) < 1$". We are also interested in the predicate $T^{f'}_K(m, r)$ where f' is the derivative of f. If f is understood, we write $T_K(m, r)$ and $T'_K(m, r)$ instead of $T^f_K(m, r)$ and $T^{f'}_K(m, r)$. For the T-predicates, circular geometry is more natural than box geometry. Let $D(m, r)$ denote the disk centered at m with radius r. It is clear that $T_1(m, r)$ is an exclusion predicate for the disc $D(m, r)$; it is also shown that if $T'_{\sqrt{2}}(m, r)$ holds then B has at most one root; therefore we use this as an inclusion predicate. But to have a confirmation predicate, we need to ensure that there is at least one root in B. For this purpose, the following 8**-point test** was developed by Sagraloff-Yap: consider 8 compass points $m + 4re^{\mathbf{i}j\pi/4}$ for $j = 0, \ldots, 7$ on the boundary of $D(m, 4r)$ (not $D(m, r)$). We give them the conventional names N, S, E, W, NE, NW, SE, SW, as shown in Figure 4.

They define 8 arcs $A_0, \ldots, A_7$ where $A_j = \{u(m + 4re^{\mathbf{i}\theta}) : j\pi/4 \leq \theta < (j + 1)\pi/4\}$. We say there is an (arcwise) u**-crossing** of A_i if the value of u at the endpoints of A_i changes sign. For instance, the endpoints of A_1 is N and NE, and there is a u-crossing at A_1 if $u(N)u(NE) < 0$. If

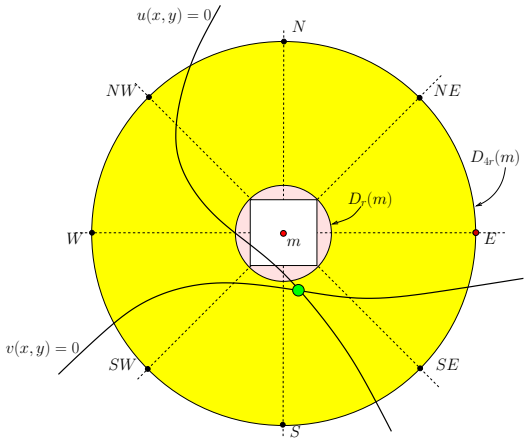


Figure 4: 8 compass points on D_{4r}.

there are exactly two u-crossings at A_j and A_k, there are exactly two v-crossings at $A_{j'}$ and $A_{k'}$, and these crossing interleave (i.e., either $j < j' < k < k'$ or $j' < j < k' < k$), then we say the 8-**point test passes** for $D(m, 4r)$; otherwise the test fails. The key result is this:

THEOREM 1 (SAGRALOFF-YAP). *Suppose that the predicate $T_6'(m, 4r)$ holds, and the 8-point test is applied to the disc $D(m, 4r)$.*

(i) If the test fails, then $D(m, r)$ is not isolating.

(ii) If the test passes, the $D(m, 4r)$ is isolating.

By choosing $m = m(B)$ and r to be the radius of box B, this test allows us to reject B (if the test fails) or to accept a larger disk $D(m, 4r)$. There is an issue of exactness of this test: although the four cardinal points (N, S, E, W) are dyadic, the other four ordinal points (NE, NW, SE, SW) are irrational. It is shown [29] that the 8-point test remains valid when we approximate the ordinal points by other points on the boundary of $D(m, 4r)$, provided their angular deviation from the perfect positions is less than 2.5°. In particular, we can use Pythagorean triples to provide rational approximations for NE, NW, SE, SW. Such approximate points lie exactly on the boundary of $D(m, 4r)$ and can be chosen arbitrarily close to NE, NW, SE, SW. The simplest Pythagorean triple that is sufficiently close is $(20, 21, 29)$, with $\arcsin(20/29) \approx 43.60^\circ$. A more accurate triple is $(119, 120, 169)$ with $\arcsin(119/169) = 44.76^\circ$. Using such triples, the 8-point test can be implemented without error using rational arithmetic (not BigFloats). There are several ways to implement `Ceval`: let m, r denote the midpoint and radius of B. The exclusion predicate is given by $C_{out}(B) \equiv T_1(m, r)$. The inclusion predicate $C_{in}(B)$ can be defined be the conjunction of $T_6'(m, 4r)$, $T_{\sqrt{2}}'(m, 8r)$, and the passing of the 8-point test on $D(m, 4r)$. Yakoubsohn's algorithm is much simpler: it has the same exclusion predicate, but the inclusion predicate is that $r < \varepsilon$ (for some $\varepsilon > 0$). The requirement $T_{\sqrt{2}}'(m, 8r)$ ensures that if two confirmed discs intersect, we can discard any one of them. Thus we output exactly one isolating disk per root.

Let us first look at the raw performance of `Ceval` in Table 3. For each test polynomial, we have two experiments: first to isolate all the roots in the box $[-2, 2] \times [-2, 2] \subseteq \mathbb{C}$, and next to isolate all complex roots. In the latter case, we use the usual Cauchy bound (basically maximum modulus of all the coefficients) to determine a box containing all roots. As expected, the latter takes more iterations and more time. We do not need a head-to-head comparison between `Ceval` with the Newton-type algorithms because `Ceval` is 3 orders of magnitude faster.

Polynomial	$[-2,2] \times [-2,2]$		All Roots	
	Iters	Time(ms)	Iters	Time(ms)
random10	1909	1.741	2615	2.342
nroots10	2037	1.838	2037	1.816
chebyshev20	12805	26.262	18533	39.821
nroots20	7989	16.593	7989	16.444
laguerre20	805	1.747	38253	79.055
hermite20	1685	3.680	17093	35.618
wilk20	581	1.371	40589	97.393
chrma22	4949	10.978	36749	80.626
chrmc23	7101	16.840	43389	107.063
random30	16013	62.602	27413	110.893
random40	27419	178.732	45722	315.381
random50	43160	427.576	71905	743.922
random60	60757	843.580	107746	1495.772
random70	80215	1481.286	108310	2104.210
random80	111795	2685.381	121129	2946.031
random90	139605	4211.166	221837	6789.341

Table 3: `Ceval` algorithm on $B_0 = [-2, 2] \times [-2, 2]$. Runtimes are in milli seconds.

We now turn our attention to the comparison of `Ceval` with Yakoubsohn's method. We need to choose a fixed ε: one natural choice is to use the standard root separation bound estimate. But this choice would really slow down Yakoubsohn's algorithm. For our timings, we therefore choose a fixed value of ε depending on the Core Level. At Level 1, we choose $\varepsilon = 0.0001$. The performance comparison between these two approaches can be found in Table 4. As expected, Yakoubsohn's method is faster since it does not bear the burden of confirming roots. However, in our tests the `Ceval` algorithm always operates on a fewer number of boxes because we can stop subdivision once a box satisfies our inclusion predicate. Column 2 in Table 4 shows the number of output boxes from Yakoubsohn's algorithm for each test polynomial. For instance, the first entry is a random polynomial of degree 10, but 76 boxes are output. We expect that at least 66 of these boxes are spurious (assuming no roots lie on the boundary of a box). In our tests, his algorithm tend to produce an average of 8 output boxes around each root. This may be seen in Table 4 by dividing the number of output boxes by the degree of the polynomial.

Finally, we compare `Ceval` with the `MPSolve` package. which is based on the Aberth-Erlich simultaneous iteration. The timings for `MPSolve` were generated using the UNIX `time` command and are not as accurate as the timings generated from our code and are provided as a rough indicator of performance. The list of timings can be found in Table 5.

`Ceval` performs about the same as `MPSolve` for polynomials with degree $n < 30$. For higher degree polynomials, their performance starts to diverge with `MPSolve` being consistently faster. At $n = 40$ we see that `Ceval` is generally up to five or eight times slower. One of the factors that

	Boxes	Iters		Time(ms)	
	(Yako.)	Ceval	Yako.	Ceval	Yako.
random10	76	2615	3105	2.721	2.282
nroots10	96	2037	2581	1.773	1.802
chebysh20	176	18533	19509	38.462	35.544
nroots20	192	7989	8885	16.481	14.526
laguerr20	192	38253	39845	77.991	65.489
hermite20	160	17093	18309	35.690	29.955
wilk20	128	40589	41909	84.321	69.130
random20	147	8201	8985	19.750	14.805
chrma22	168	36749	37661	83.140	67.142
chrmc23	200	43389	43813	101.816	83.464
random30	228	27413	28441	123.294	89.068
random40	296	45722	46957	337.769	245.090
random50	361	71905	73347	783.600	578.127
random60	426	107746	109317	1646.602	1191.557
random70	511	108310	110064	2213.269	1623.679
random80	581	121129	122990	3021.738	2347.435
random90	665	221837	223842	6839.414	5385.514

Table 4: Comparison of Ceval and Yakoubsohn's method with $\varepsilon = 0.0001$. Observe that Yakoubsohn's method is always faster, yet Ceval is comparable because it always operates on a fewer number of boxes.

works against Ceval is the estimate of B_0 from the Cauchy bound. For instance, the Wilkinson's degree 40 polynomial has an estimated $B_0 = [-2048, 2048] \times [-2048, 2048]$, which is much larger than the minimal bound of 40. The number of iterations of the Aberth-Erlich iteration that are required to converge to a root is not directly related to the root bounds, or to the distribution of roots. The behavior of this iteration (and the reason for its seemingly wonderful convergence properties) is not well understood; a discussion can be found in [2].

	Time(ms)	
	Ceval	MPSolve
chebyshev20	39.821	27
laguerre20	79.055	87
hermite20	35.618	81
wilk20	97.393	49
chrma22	80.626	47
chrmc23	107.063	61
hermite40	525.80	88
wilk40	1142.82	153

Table 5: Comparison of Ceval and MPSolve.

We must keep in mind that the Aberth-Erlich (or indeed the Weierstrass-Durand-Kerner) iteration does not output a list of isolating boxes, rather just approximations of roots. We have no guarantee that the iteration will converge to "reasonable" approximations. Additionally, the strength of subdivision based methods is that they can operate on tight areas of interest while the simultaneous iteration based methods necessarily have to approximate all roots. Consequently, we cannot provide a comparison of the two methods operating on such a preselected area because the MPSolve software [3] does not support such an option.

7. EXPERIMENTS AT LEVEL 2

We now provide a comparison of running times of our algorithms at Core Level 2. Basically, machine double are now replaced by BigFloat numbers In terms of implementation, this comes almost for free because of Core Library's ability to reuse the same program, just at the cost of re-compilation.

The performance of the three Newton-type operators is presented in Table 6. As in the case of Level 1, there is no significant difference between the three operators. At Level 2 however, the interval Newton operator appears to be faster than the Krawczyk operator by a small but consistent margin. Given that arithmetic operations are more expensive at this Level, the method that processes the fewest boxes is bound to be faster. Also, note that the Hansen-Sengupta operator lags further behind the other two as a result of its expensive extended interval arithmetic computations.

These tests are carried out on randomly generated polynomials of the degrees shown. These have been constructed densely, with each coefficient being separately generated.

Deg	Iters			Time(ms)		
	HS	N	K	HS	N	K
4	325	333	357	0.320	0.334	0.463
6	877	805	829	2.285	1.769	2.164
8	1557	1317	1349	6.972	5.633	6.559
10	2461	2093	2069	19.066	15.730	17.410
12	3445	2965	2965	41.488	35.037	39.390
14	4677	3997	4021	87.262	71.523	77.699

Table 6: Comparison of Newton type operators at Level 2 on $[-2, 2] \times [-2, 2]$

As in the case of Level 1, Ceval continues to be three orders of magnitude faster than the Newton type operators. The results make up Table 7.

Deg	Output	Iters		Time(ms)	
	(Yako.)	Ceval	Yako.	Ceval	Yako.
10	72	1965	2381	0.801	0.876
20	128	7909	8565	12.587	12.257
30	224	16413	17381	65.162	53.582
40	248	29293	30405	203.899	163.549

Table 7: Comparison Ceval with Yakoubsohn's pure exclusion approach over $[-2, 2] \times [-2, 2]$

The results of this section show that the performance of our algorithms at Level 2 is between 20 and 50 times slower than at Level 1. This is an expected consequence of using extended precision types. To improve performance, it might be advantageous to carry out as many operations as possible at machine precision, and to control precision growth in areas that require it.

The Ceval implementation is currently not capable of switching Core levels at run time for portions of the working set; it must run entirely at Level 1 or Level 2, a decision made at compile time. This is one aspect of our implementation that we plan to improve.

8. CONCLUSION AND FUTURE WORK

Our experimental results show that the interval arithmetic based approaches suffer from a serious degradation in performance as the degree n of the polynomial increases. Some of this degradation in performance can be attributed to the overestimation of function range by interval extensions, but methods to compensate for it tend to be computationally expensive. Further, these operators suffer from issues due to roots that lie on box boundaries. Overall, this approach appears to suffer from various practical and performance issues, and its use cannot be recommended.

Our experimental results for the `Ceval` algorithm appear quite encouraging. It is efficient and robust, and works well on a large range of polynomials, of various degrees and with densely generated random coefficients. It provides stronger guarantees than Yakoubsohn's exclusion based approach at a comparable speed. Further, both of these approaches perform three orders of magnitude faster than the interval arithmetic based approaches. However, the performance of the predicate T_K breaks down due to the growth of $n!$ for polynomials of degree $n > 90$. We plan to extend the range of the achievable degrees in Level 1 (see below). But even without these extensions, we believe that the algorithm is an efficient and viable choice for isolating roots of complex polynomials of degree $n < 90$. The following plans for future work are related to the development of `Ceval`-like algorithms.

- We noted that the latest version of the `Ceval` paper (to appear in ISSAC 2011) provided a simplified alternative to the 8-point test. Recall that the success of the predicate $T_1(m, r)$ implies that there is no roots in the disk $D(m, r)$ (see Section 6). The simplification is based on the observation that the failure of $T_1(m, r)$ could serve as an inclusion predicate, albeit for a much larger disk: $D(m, 2nr)$. Both versions have the same asymptotic bound of $\widetilde{O}(n^4L^2)$ bit complexity for the benchmark problem. Since their relative performance in practice is unclear, the full version of this paper will compare these two versions.

- As shown above, just switching from Level 1 to Level 2 causes our algorithms to slow down by a factor of up to 20 to 50. We would like to explore a fixed precision BigFloat that behaves closer to a machine type, but with larger number of bits of precision.

- The 8-point-test evaluates the input polynomial at ordinal compass points using the `BigRat` type. Since the `BigRat` type is represented by a pair of `BigInt` instances (a, b) for $\frac{a}{b}$, the exponentiation of these values to a high power will be slow and the representation will be expensive in terms of memory usage as well. Also, since the `BigRat` representation must always contain no common factors, many expensive GCD operations are required as well. One possible change in this area is to use rounded interval arithmetic to estimate the range of f over the interval lower and upper bound of the approximate value of the ordinal points. Clearly, if the interval is either entirely negative or positive, then so is the sign of f at the exact ordinal point.

- We need to explore approaches that allow T_K to remain performant for higher degree polynomials as well. Some of the directions we can look in include the truncated evaluation of the Taylor series and controlled precision growth (with correct rounding) of our calculations.

- Note that our implementation runs either entirely at Level 1 or entirely at Level 2. It would be desirable to implement a wrapper over a machine precision type that can detect underflows and overflows, and switch calculation over to Level 2. We are currently working on the outline of such a type, but a lot of work remains to be done to test it and integrate it with our `Ceval` implementation.

- Recall that Sturm and Descartes' methods [22, 29] are subdivision methods for real roots. It is possible to generalize both to real solutions of bivariate systems. Their performance against evaluation methods should be of interest.

9. REFERENCES

[1] E. Berberich, P. Emeliyanenko, and M. Sagraloff. An elimination method for solving bivariate polynomial systems: Eliminating the usual drawbacks. In Workshop on Algorithm Engineering and Experiments (ALENEX11), 2011. Jan 22, 2011. San Francisco, California.

[2] D. A. Bini. Numerical computation of polynomial zeroes by means of Aberth's method. Numerical Algorithms, 13:179–200, 1996.

[3] D. A. Bini and G. Fiorentino. MPSolve: manual for the MPSolve package. `http://www.dm.unipi.it/cluster-pages/mpsolve/mpsolve.pdf`.

[4] D. A. Bini and G. Fiorentino. Design, analysis, and implementation of a multiprecision , polynomial rootfinder. Numerical Algorithms, 23:127âĂŞ173, 2000.

[5] M. Burr, F. Krahmer, and C. Yap. Continuous amortization: A non-probabilistic adaptive analysis technique. Electronic Colloquium on Computational Complexity (ECCC), TR09(136), December 2009.

[6] M. Burr, V. Sharma, and C. Yap. Evaluation-based root isolation, 2011. In preparation.

[7] J. Cheng, S. Lazard, L. Peñaranda, M. Pouget, F. Rouillier, and E. Tsigaridas. On the topology of planar algebraic curves. In Proc. 25th Symp. on Comp. Geom. (SoCG'09), pages 361–370, 2009.

[8] J.-S. Cheng, X.-S. Gao, and J. Li. Root isolation for bivariate polynomial systems with local generic position method. Mm research preprints, KLMM, Chinese Academy of Sciences, 2008.

[9] Core Library homepage, since 1999. Software download, source, documentation and links: `http://cs.nyu.edu/exact/core/`.

[10] J.-P. Dedieu and J.-C. Yakoubsohn. Localization of an algebraic hypersurface by the exclusion algorithm. Applicable Algbebra in Engineering, Communication and Computing, 2:239–256, 1992.

[11] D. Diochnos, I. Emiris, and E. Tsigaridas. On the complexity of real solving bivariate systems. J. Symbolic Computation, 44:818âĂŞ835, 2009.

[12] E. R. Hansen. On solving systems of equations using interval arithmetic. Math. Comp., 22, 1968.

[13] E. R. Hansen. A globally convergent interval method for computing and bounding real roots. BIT, 16:415–424, 1978.

[14] N. Kamath. Subdivision algorithms for complex root isolation: Empirical comparisons. Master's thesis, Oxford University, Oxford Computing Laboratory, Aug. 2010.

[15] R. Krawczyk. Newton-Algorithmen zur Bestimmung von Nullstellen mit Fehlerschranken. Computing, 4:187–201, 1969.

[16] L. Lin and C. Yap. Adaptive isotopic approximation of nonsingular curves: the parameterizability and nonlocal isotopy approach. Discrete and Comp. Geom., 45(4):760–795, 2011.

[17] D. P. Mitchell. Robust ray intersection with interval arithmetic. In Graphics Interface'90, pages 68–74, 1990.

[18] R. Moore and S. Jones. Safe starting regions for iterative methods. SIAM J. Num.Analysis, 14(6):1051–1065, 1977.

[19] R. E. Moore. Interval Analysis. Prentice Hall, Englewood Cliffs, NJ, 1966.

[20] R. E. Moore. A test for existence of solution to nonlinear systems. SIAM J. Numer. Anal., 14:611–615, 1977.

[21] R. E. Moore and L. Qi. A successive interval test for nonlinear systems. SIAM J. Numer. Anal., 19:845–850, 1982.

[22] B. Mourrain, F. Rouillier, and M.-F. Roy. The Bernstein basis and real root isolation. In J. E. Goodman, J. Pach, and E. Welzl, editors, Combinatorial and Computational Geometry, number 52 in MSRI Publications, pages 459–478. Cambridge University Press, 2005.

[23] A. Neumaier. Interval Methods for Systems of Equations. Cambridge University Press, Cambridge, 1990.

[24] K. Nickel. On the Newton Method in Interval Analysis. Research Memorandum MRC Technical Summary Report #1136, University of Wisconsin, Madison, 1971.

[25] V. Y. Pan. Solving a polynomial equation: some history and recent progress. SIAM Review, 39(2):187–220, 1997.

[26] S. Plantinga and G. Vegter. Isotopic approximation of implicit curves and surfaces. In Proc. Eurographics Symposium on Geometry Processing, pages 245–254, New York, 2004. ACM Press.

[27] H. Ratschek and J. Rokne. Computer Methods for the Range of Functions. Horwood Publishing Limited, Chichester, West Sussex, UK, 1984.

[28] F. Rouillier and P. Zimmermann. Efficient isolation of a polynomial real roots. J. Comp. and Appl. Math., 162:33–50, 2003.

[29] M. Sagraloff and C. K. Yap. A simple but exact and efficient algorithm for complex root isolation. In 36th Int'l Symp.Symbolic and Alge.Comp. (ISSAC 2011), 2011. To Appear. June 8-11, San Jose, California.

[30] A. Schönhage. The fundamental theorem of algebra in terms of computational complexity, 1982. Manuscript , Department of Mathematics, University of Tübingen. Updated 2004.

[31] S. Smale. The fundamental theorem of algebra and complexity theory. Bulletin (N.S.) of the AMS, 4(1):1–36, 1981.

[32] J. M. Snyder. Interval analysis for computer graphics. SIGGRAPH Comput.Graphics, 26(2):121–130, 1992.

[33] V. Stahl. Interval Methods for Bounding the Range of Polynomials and Solving Systems of Nonlinear Equations. Ph.D. thesis, Johannes Kepler University, Linz, 1995.

[34] G. Taubin. Rasterizing algebraic curves and surfaces. IEEE Computer Graphics and Applications, 14(2):14–23, 1994.

[35] The FRISCO Consortium. FRISCO Polynomial test suite. `http://www-sop.inria.fr/saga/POL/`.

[36] J.-C. Yakoubsohn. Numerical analysis of a bisection-exclusion method to find zeros of univariate analytic functions. J. of Complexity, 21:652–690, 2005.

[37] C. K. Yap. Tutorial: Exact numerical computation in algebra and geometry. In Proc. 34th Int'l Symp. Symbolic and Algebraic Comp. (ISSAC'09), pages 387–388, 2009. KIAS, Seoul, Korea, Jul 28-31, 2009.

[38] J. Yu, C. Yap, Z. Du, S. Pion, and H. Bronnimann. Core 2: A library for Exact Numeric Computation in Geometry and Algebra. In 3rd Proc. Int'l Congress on Mathematical Software (ICMS), pages 121–141. Springer, 2010. LNCS No. 6327.