Towards Soft Exact Computation*

Chee Yap

Department of Computer Science
Courant Institute, NYU
New York, NY 10012, USA
{yap}@cs.nyu.edu

Abstract. Exact geometric computation (EGC) is a general approach for achieving robust numerical algorithms that satisfy geometric constraints. At the heart of EGC are various Zero Problems, some of which are not-known to be decidable and others have high computational complexity. Our current goal is to introduce notions of "soft-ε correctness" in order to avoid Zero Problems. We give a bird's eye view of our recent work with collaborators in two principle areas: computing zero sets and robot path planning. They share a common Subdivision Framework. Such algorithms (a) have adaptive complexity, (b) are practical, and (c) are effective. Here, "effective algorithm" means it is easily and correctly implementable from standardized algorithmic components. Our goals are to outline these components and to suggest new components to be developed. We discuss a systematic pathway to go from the abstract algorithmic description to an effective algorithm in the subdivision framework.

1 Introduction

We are interested in computations involving the continuum and the reals. Most algorithms in scientific computation and engineering are of this nature (e.g., [41]). In practice, they fall under the domain of numerical computing [49,28]. Numerical algorithms are expected to make errors and the question of their correctness takes on a much more subtle meaning than the typical discrete or algebraic algorithms. One way to avoid these errors is to reformulate these problems algebraically with exact algorithms. This is often possible but not always desirable or practical [58]. So we aim at solutions that are fundamentally numerical.

The most widely used procedure for constructing numerical algorithms is to first construct an algorithm, say A, based on a real RAM computational model ([51, Section 9.7] or [40]) and then implement A as an algorithm $\widetilde{A}$ of the Standard Model of Numerical Analysis ([21, Section 2.2] or [48]). All operations in A are exact, but in $\widetilde{A}$, each numerical operation $x \circ y$ is replaced by an approximation $x \,\widetilde{\circ}\, y$ whose relative error is at most $u > 0$ (the unit round-off error). In principle, such an $\widetilde{A}$ (unlike A) can be implemented on a Turing machine. For simplicity, we assume no overflow in the Standard Model. In the simplest case, $\widetilde{A}$ is just a copy of A except for the $\circ \mapsto \widetilde{\circ}$ transformation. In the analysis of algorithms, our first task is to prove the correctness of a given algorithm. In the present setting, we are faced with a pair $(A, \widetilde{A})$ of algorithms. The modus operandi is to (i) show correctness of A and (ii) do error analysis of $\widetilde{A}$. There are issues with this procedure. It is the correctness of $\widetilde{A}$ that we need. Correctness of A is a necessary, but not sufficient condition. The translation $A \mapsto \widetilde{A}$ hits a snag if there is branch-at-zero step in A: we must decide if a pivot value is exactly zero. This situation arises, for instance, in Gaussian elimination with partial pivoting (GEPP). In this paper, the problem of deciding if a numerical value x is equal to zero will be called "the Zero Problem". In reality, there are various Zero Problems – see [46] for a formal definition of these problems. When x is algebraic, the Zero Problem is decidable, but otherwise, it is generally not-known to be decidable. Partial solutions include replacing the standard model by arbitrary precision arithmetic ("BigNums"), or using the modified statement: "$\widetilde{A}$ is correct if u is small enough." What is the status of $\widetilde{A}$ if u is not small enough? There is also no guarantee that such a u exists. Basically, there are still Zero Problems lurking beneath such reformulations.

In this paper, we wish to avoid the Zero Problems by modifying the notion of correctness of the given computational problem P: instead of seeking algorithms that are (unconditionally) correct, we seek algorithms that are **ε-correct** where $\varepsilon > 0$ is an extra input, called the **resolution parameter**. Let P_ε denote

* This work is supported by NSF Grants CCF-1423228 and CCF-1564132.

this modified problem. As $\varepsilon \to 0$, then P_ε converges to the original problem. Unlike the "correct when u is small enough" criteria, we want our ε-correctness criterion to be met for each $\varepsilon > 0$. What we seek is an algorithm for P_ε that is **uniform** in ε. In discrete optimization algorithms, this is called an "approximation scheme" [44]. The polynomial-time versions of such schemes are called PTAS ("polynomial-time approximation schemes"). It is known that unless $P = NP$, the "hardest" problems in the complexity class APX do not have PTAS's. Although our continuum problems do not fall under such discrete complexity classes, our Zero Problems represent fundamental intractability analogous to the $P = NP$ barrier. The difference is that discrete intractability leads to exhaustive or exponential search, but continuum intractability leads to a halting problem.

Besides the viewpoint of combinatorial optimization, we briefly note other ways of using ε in the literature. In numerical computation, ε is commonly interpreted as an *a priori* guaranteed upper bound on the forward and/or backwards error of the algorithm's output. Depending on whether the error is taken in the absolute or relative sense of error, this gives at least 6 distinct notions of "ε-correct". As in the above Standard Model (with unit roundoff error u) such interpretations do not automatically escape the Zero Problem. Such interpretations of ε may be extended to geometry. For example, instead of bound on numerical errors, we interpret ε as bound on deviations from ideal geometric objects such as points, curves or surfaces. Suppose the output of the algorithm is a finite set $S = \{p_1, \ldots, p_n\}$ of points (e.g., S are the extreme points of a convex hull), one might define S to be ε-correct if each $p_i \in B_\varepsilon(p_i^*)$ where $S^* = \{p_1^*, \ldots, p_n^*\}$ is the exact solution. Here, $B_\varepsilon(p)$ denotes the ball centered at p of radius ε. Unfortunately, such a view still encodes a (deferred) Zero Problem because the condition $p_i \in B_\varepsilon(p_i^*)$ is a "hard predicate". Intuitively, the ball $B_\varepsilon(p_i)$ has a hard boundary (this is a "hard-ε"). Our goal is to soften such boundaries, using suitable **soft-ε** criteria. This will be illustrated through some non-trivial problems.

In this paper, we give a bird's eye view of a collection of papers over the last decade with our collaborators, from computing zero sets to path planning in robotics. We will attempt to put them all under a single subdivision rubric. We are less interested in the specific results or algorithms than in the conceptual framework they suggest.

1.1 From Zero Problems to Predicates

The "soft exact computation" in our title is an apparent oxymoron since softness suggests numerical approximation in opposition to exact computation. The notion of "exactness" here comes from computational geometry [8] where it is assumed that algorithms must compute geometric objects with the exact combinatorial or topological structure. The most successful way to achieve such algorithms is called "Exact Geometric Computation" (EGC) [46]. In EGC, we explicitly reduce our computation to various Zero Problems. An example of a Zero Problem is to decide if a determinant $D(\boldsymbol{x})$ is zero where $\boldsymbol{x}$ are the entries of a $n \times n$ matrix. This may arise as the so-called orientation predicate in which $\boldsymbol{x}$ represent n vectors of the form $\boldsymbol{a}_i - \boldsymbol{a}_0$ $(i = 1, \ldots, n)$ with $\boldsymbol{a}_j \in \mathbb{R}^n$ $(j = 0, \ldots, n)$. For real geometry, we usually need a bit (sic) more than just deciding zero or not-zero: we need $\texttt{sign}(D(\boldsymbol{x})) \in \{-1, 0, +1\}$. If this sign computation is error-free, then the combinatorial structure is guaranteed to be exact. Practitioners avoid the Zero Problem by defining an **approximate sign function**, $\widetilde{\texttt{sign}}(D(\boldsymbol{a})) \in \{-1, \widetilde{0}, 1\}$ where the "approximate zero sign" $\widetilde{0}$ is determined by the condition $|D(\boldsymbol{a})| < \varepsilon$. This is called "$\varepsilon$-tweaking" (using different multiples of ε's in different parts of the code) to reduce the possibility of failure. This tweaking is rarely justified (presumably it introduces some ε-correctness criteria, but what is it?)

What is the correct way to use this ε? We need a different perspective on Zero Problems: each Zero Problem arises from the evaluation of a predicate. We distinguish two kinds of predicates: **logical predicates** are 2-valued (`true` or `false`) but **geometric predicates** are typically 3-valued $(-1, 0, +1)$. Thus we view $\texttt{sign}(D(\boldsymbol{x}))$ above as a geometric predicate. Calling the sign function a "predicate" imbues it with geometric meaning: thus when we call $\texttt{sign}(D(\boldsymbol{x}))$ an orientation predicate, we know that we are dealing with a geometrically meaningful property of the n vectors arising from $n + 1$ points. These 3 sign values are not fully interchangeable: we call 0 the **indefinite value** and the other two values are **definite values**.

To continue this discussion, let us fix a geometric predicate C on $\mathbb{R}^m$,

$$C : \mathbb{R}^m \to \{-1, 0, +1\}\,. \tag{1}$$

We assume C has[1] the **Bolzano property** in the sense that if S is a connected set and there exists $\boldsymbol{a}, \boldsymbol{b} \in S$ such that $C(\boldsymbol{a}) = -1$ and $C(\boldsymbol{b}) = +1$ then there exists $\boldsymbol{c} \in S$ such that $C(\boldsymbol{c}) = 0$. For example, if $C(\boldsymbol{a}) := \mathtt{sign}(D(\boldsymbol{a}))$ where $D(\boldsymbol{x})$ is the above determinant (with $m = n^2$), then C has the Bolzano property. More generally, if $D(\boldsymbol{x})$ is any continuous function, its sign predicate is Bolzano.

Next we take a critical step by extending the predicate C on points $\boldsymbol{a} \in \mathbb{R}^m$ to sets of points $S \subseteq \mathbb{R}^m$. The significance is that we have moved from algebra to analysis: we could treat $D(\boldsymbol{a})$ algebraically since it amounts to polynomial evaluation, but the analytic properties come to the forefront when we consider the set $D(S)$. And soft-ε concepts are fundamentally analytic.

We define the **set extension** of C as follows: for $S \subseteq \mathbb{R}^m$, define $C(S) = 0$ if there exists $\boldsymbol{a} \in S$ such that $C(\boldsymbol{a}) = 0$; otherwise, $C(S)$ may be defined to be $C(\boldsymbol{a})$ for any $\boldsymbol{a} \in S$. The Bolzano property implies that $C(\boldsymbol{a})$ is well-defined. Thus the set extension of C is the predicate $C : 2^{\mathbb{R}^m} \to \{-1, 0, 1\}$ where 2^X denotes the power set of any set X. If we are serious about computation, we know that S must be suitably restricted to "nice" subsets of $\mathbb{R}^m$. Following the lead of interval analysis [34], we interpret "nice" to mean axes-aligned full-dimensional boxes in $\mathbb{R}^m$. Let $\square\mathbb{R}^m$ be the collection of such boxes. When the domain of the set extension of C is restricted to such boxes, we have this **box predicate**

$$C : \square\mathbb{R}^m \to \{-1, 0, +1\} \tag{2}$$

where the symbol 'C' from the point predicate in (1) is reused. This reuse is justified if we regard $\mathbb{R}^m$ as a subset of $\square\mathbb{R}^m$. In other words, each element of $\square\mathbb{R}^m$ is either a full-dimensional box or a point.

Our next goal is to approximate the box predicate C. Consider another box predicate

$$\widetilde{C} : \square\mathbb{R}^m \to \left\{-1, \widetilde{0}, +1\right\}. \tag{3}$$

Call $\widetilde{C}$ a **soft version** of (2) if it is conservative and convergent: **conservative** means $\widetilde{C}(B) \neq \widetilde{0}$ implies $\widetilde{C}(B) = C(B)$; **convergent** means if $\{B_i : i \geq 0\}$ is an infinite monotone[2] sequence of boxes that converges to a point $\boldsymbol{a}$, then $\widetilde{C}(B_i)$ converges to $C(\boldsymbol{a})$, i.e., $\widetilde{C}(B_i) = C(\boldsymbol{a})$ for i large enough. We say $\{B_i : i \geq 0\}$ is **firmly convergent** if there is some $\sigma > 0$ such that $B_{i+1} \subseteq B_i/\sigma$ for all $i \geq 0$. We say $\widetilde{C}$ is **firm** relative to C if $C(B) \neq 0$ implies $\widetilde{C}(B/2) \neq \widetilde{0}$. The "2" in this definition may be replaced by any **firmness factor** $\sigma > 1$, if desired. As $\sigma \to 1$, the computational cost of C would increases. For resolution-exact path planning, we only need[3] half of the properties of firmness, namely, $C(B) = 1$ implies $\widetilde{C}(B/\sigma) = 1$ [50]. But even path planning may exploit the other half of firmness (i.e., $C(B) = 0$ implies $\widetilde{C}(B/\sigma) = 0$) because it could lead to faster determination of NO-PATH.

From any geometric predicate C, we derive three logical predicates C_+, C_-, C_0 in a natural way: $C_+(\boldsymbol{a}) := [[C(\boldsymbol{a}) > 0]]$. The notation "$[[S]]$" denotes the truth value of any sentence S: for instance $[[1 > 2]]$ is equal to `false` but $[[1 + 2 = 3]]$ is equal to `true`. We call "$[[S]]$" a **test**. In general, $S = S(\boldsymbol{x})$ depends on variables $\boldsymbol{x}$, and our test $[[S(\boldsymbol{x})]]$ represents a logical function. Similarly, $C_-(\boldsymbol{a}) := [[C(\boldsymbol{a}) < 0]]$ and $C_0(\boldsymbol{a}) := [[C(\boldsymbol{a}) \neq 0]]$. This last predicate is called the **exclusion predicate** and is very important for us: it is used in all of our algorithms. Again we extend these predicates naturally to sets or boxes as above. In particular, we have $C_0(B) = [[0 \notin C(B)]]$.

2 Two Illustrative Classes of Problems

We introduce two classes of geometric problems to serve as running examples:

(A) Computing a Zero Set $\mathtt{Zero}(\boldsymbol{f})$.
Here, $\boldsymbol{f} = (f_1, \ldots, f_m)$, $f_i : \mathbb{R}^n \to \mathbb{R}$, and $\mathtt{Zero}(\boldsymbol{f}) := \{\boldsymbol{a} \in \mathbb{R}^n : f_i(\boldsymbol{a}) = 0, i = 1, \ldots, m\}$. We can also define this problem for complex zeros, i.e., $\mathtt{Zero}(\boldsymbol{f}) \subseteq \mathbb{C}^n$. In the case f_i are integer polynomials, $\mathtt{Zero}(\boldsymbol{f})$

[1] After Bernard Bolzano (1817). Bolzano's Theorem states that if $a < b$ and $\mathtt{sign}(f(a)f(b)) < 0$ then there is some $c \in (a, b)$ such that $f(c) = 0$. See also [43,3] for this principle in real root isolation.

[2] Monotone means $B_{i+1} \subseteq B_i$ for all i.

[3] The factor $\sigma > 1$ was call the "effectivity factor" in [50]. In the present paper, we avoid this terminology since it conflicts with our notion of "effectivity" of this paper.

is an algebraic variety where an exact algebraic solution is often interpreted to mean computing some nice representative (e.g., a Gröbner basis) of the ideal generated by $\boldsymbol{f}$. But we are literally interested in the continuum: we seek some "explicit" representation of $\texttt{Zero}(\boldsymbol{f})$ as a subset of $\mathbb{R}^n$. Invariably, "explicit" has to be numerical, not symbolic. For example, an explicit solution to $\texttt{Zero}(f_1)$ where $f_1(x) = x^2 - 2$ may be 1.4 but not the expression "$\sqrt{2}$". For our discussion, let us interpret an explicit representation to mean a simplicial complex K [9, Chap.7] of the same dimension as $\texttt{Zero}(\boldsymbol{f})$, and whose support $\underline{K} \subseteq \mathbb{R}^n$ is ε-isotopic to $\texttt{Zero}(\boldsymbol{f})$. This definition implies that their Hausdorff distance satisfies $d_H(\underline{K}, \texttt{Zero}(\boldsymbol{f})) < \varepsilon$, but more is needed: an ε-isotopy maps points in $\texttt{Zero}(\boldsymbol{f})$ to points in $\underline{K}$ within a distance ε. It is important that K has the same dimension as $\texttt{Zero}(\boldsymbol{f})$: for instance, if $\texttt{Zero}(\boldsymbol{f})$ is a curve in $\mathbb{R}^3$, we really want the output K to represent a polygonal curve $\underline{K}$. In contrast, a common output criteria asks for an ε-tubular path containing $\texttt{Zero}(\boldsymbol{f})$. Unfortunately, this allows the curve to have unexpected behavior within the tube (e.g., doubling back arbitrarily far on itself within the tube). Most of the current research are aimed at cases where $\texttt{Zero}(\boldsymbol{f})$ is zero-dimensional (finite set) or co-dimension one (hypersurface). We mostly focus on the zero-dimensional case in this survey.

(B) Robot path planning.
Suppose a robot R_0 is fixed. Then the problem is: given a polyhedral obstacle set $\Omega \subseteq \mathbb{R}^k$ $(k = 2, 3)$ and start α and goal β configurations, find an Ω-avoiding path π of R_0 from α to β; or declare NO-PATH if such π's do not exist. Let $\mathcal{C}space = \mathcal{C}space(R_0)$ denote the configuration space, and $\mathcal{C}free = \mathcal{C}free(R_0, \Omega)$ denote the Ω-free configurations in $\mathcal{C}space$. For instance, if R_0 is a rigid spatial robot, then configurations are elements of $SE(3) = \mathbb{R}^3 \times SO(3)$ where $SO(3)$ are the orthogonal 3×3 matrices representing rotations. One challenge in this area is to produce implemented algorithms that are rigorous yet competitive with the practical approaches based on sampling.

Both these problems have large literatures. Problem (A) is a highly classical problem in mathematics with applications in geometric, numeric and symbolic computation. Problem (B) is central to robotics. In both cases, there are many available algorithms, and thus there are high standards for any proposed new algorithm, both theoretically and practically. In particular, they need to be implemented and compared to existing ones: subdivision algorithms appear to be able to meet both criteria. In robotics, to be "practical" includes an informal requirement of being "real time" for standard size input instances. In contrast, exact algorithms (especially for Problem (B)) are rarely implemented.

There are Zero Problems in both (A) and (B) as formulated above. For (A), even for the case $n = m = 1$ (univariate roots) where the input $\boldsymbol{f} = (f_1)$ is polynomial, we face Zero Problems. These Zero Problems are not an issue when f_1 has rational coefficients; but we are interested in coefficients that are algebraic numbers or number oracles [4,29]. For real roots, there are many complete real RAM algorithms based on Sturm sequences, on Descartes rule of sign or on Newton-bisection. In each case, the algorithms call for testing if $f_1(a) = 0$ for various points $a \in \mathbb{R}$ (we may assume a is a dyadic number, but this does not make the test any easier when the coefficients of f_1 are irrational). The very formulation of root isolation requires the output interval to have exactly one root, possibly a multiple root. Distinguishing between two simple roots that are close together from a single double root is again a Zero Problem. There are difficult Zero Problems in higher dimensional problems (even for hypersurfaces) that remain open: most current correctness criteria is conditioned on non-singularity of $\texttt{Zero}(\boldsymbol{f})$. For Problem (B), there is also a Zero Problem corresponding to the sharp transition from path to NO-PATH. We now introduce soft-ε criteria to circumvent these Zero Problems:

(a) For root isolation [53], we introduce the **ε-clustering problem**: given a region-of-interest $B_0 \in \square\mathbb{R}^n$ and $\varepsilon > 0$, output a set $\Delta_1, \ldots, \Delta_k \subseteq 2B_0$ of disjoint balls with radii $< \varepsilon$, and output multiplicities $\mu_1, \ldots, \mu_k$ satisfying

$$\mu_i := \#_{\boldsymbol{f}}(\Delta_i) = \#_{\boldsymbol{f}}(3\Delta_i) \geq 1$$

where $\#_{\boldsymbol{f}}(S)$ is the total multiplicities of the roots in S. The union of these balls must cover all the roots of $\boldsymbol{f}$ in B_0 but they may not include any root outside of $2B_0$. Each Δ_i represents a cluster of roots and the requirement $\#_{\boldsymbol{f}}(\Delta_i) = \#_{\boldsymbol{f}}(3\Delta_i)$ (which is our definition of "natural" clusters) can be viewed as a robustness property.

(b) For path planning [50], we say that the planner is **resolution-exact** if it satisfies two conditions:

(Path) If there is a path of clearance $K\varepsilon$, the algorithm must return some path π;

(Nopath) If there is no path of clearance ε/K. the algorithm must output NO-PATH.

Here $K > 1$ (called the **accuracy constant**) depends only on the algorithm and is independent of the input instance. The key is that (Path) and (Nopath) are not exhaustive because they do not cover input instances where the largest clearance of paths is strictly between ε/K and $K\varepsilon$. The planner may output either a path or NO-PATH in such instances. Since we require halting algorithms, the planner would produce an **indeterminate** answer in these cases. As we will see, indeterminacy (as opposed to determinacy) is a characteristic feature of soft-ε algorithms.

Based on criteria (a), we achieved the most general setting for root clustering algorithms – when the polynomials have number oracles as coefficients. Based on criteria (b), we developed and implemented path planners for various planar robots, culminating in our planners for rods and rings in 3D [22]. This is the first practical, non-heuristic algorithm for spatial robots with 5 degrees-of-freedom (DOFs). We remark that although we assumed that the robot R_0 is fixed, all our subdivision path planners can uniformly treat robots from a parametric family $R_0(p_0, \ldots, p_k)$. For instance, if R_0 is a 2-link robot, we may define $R_0(p_0, p_1, p_2)$ as the 2-link robot whose first two links have lengths p_1 and p_2, and these links have thickness p_0 (see [56]). Links are line segments, and they are thickened by forming a Minkowski sum with a ball of radius p_0. The thickness parameter is extremely useful in practice. Treating parametric families of robots is a feat that few exact algorithms are able to do; the only exception we know of is when R_0 is a ball, and here, the exact path planners based on Voronoi diagram can allow the radius of the ball as a parameter [35].

In general, besides the extra ε input, our subdivision algorithms also accept an input box B_0 called the region-of-interest (ROI), meaning that we wish to restrict the solutions to B_0. Specifying B_0 is not generally a burden, and is often a useful feature. In the case of root clustering, this meaning is clear – we must account for all the roots in B_0. There are Zero Problems associated with the boundary of B_0. To avoid this issue in root clustering, we allow the output clusters to include roots outside of B_0, but still within an expanded box $2B_0$ (or $(1+\varepsilon)B_0$ if so desired).

3 Effectivity of the Subdivision Framework

We have noted that the usual pathway to a numerical algorithm $\widetilde{A}$ is through an intermediate real RAM algorithm A. This $\widetilde{A}$ amounts to specifying a suitable precision for each arithmetic operation in A. The difficulty of this pathway is illustrated by the **benchmark problem** in root isolation: this is the problem of isolating all the roots of a univariate polynomial $p(x)$ with integer coefficients [5]. It has been known for about 30 years that there is an explicit real RAM algorithm A with transformation $A \mapsto \widetilde{A}$ such that $\widetilde{A}$ is a near-optimal algorithm for the benchmark problem. The algorithm A is from Schönhage-Pan (1981-92) [20]. Here "near-optimal" means[4] a bit complexity of $\widetilde{O}(n^2L)$ where $p(x)$ has degree n and L-bit coefficients. Although the construction of such an $\widetilde{A}$ from a real RAM A remains open, there are now several implementations of near-optimal algorithms based on subdivision, all shortly after the appearance of the subdivision algorithms [24]. See [52] for an account of this development (there are two parallel accounts, for complex roots and for real roots). We may ask why? Intuitively, it is because subdivision computation is reduced to operations on individual boxes (i.e., locally) and we can adjust the precision to increase as the box size decreases. In contrast, controlling the precision of arithmetic operations in a real RAM algorithm for some target resolution in the output appears to be hopelessly complicated at present.

There is no formal "subdivision model of computation". We intend our algorithms to be ultimately Turing-computable. So we only speak of the "subdivision approach or paradigm". Nevertheless, it is useful to introduce a **Subdivision Framework** which can be instantiated to produce many different algorithms.

In the simplest terms, we may describe it as follows: first assume that we are computing in $\mathbb{R}^m$, where $\square\mathbb{R}^m$ is the set of full-dimensional axes-aligned boxes. Let S be a subset of $\square\mathbb{R}^m$. Its **support** is the set $\underline{S} := \bigcup_{B\in S} B$. We call S a **subdivision** (of its support $\underline{S}$) if the interiors of any two boxes in S are disjoint. The subdivision process is typically controlled by two box predicates $C_0, C_1 : \square\mathbb{R}^m \to \{\texttt{true}, \texttt{false}\}$. Here C_0 is the standard **exclusion predicate**, and C_1 the **inclusion predicate** (which varies with the application).

[4] The $\widetilde{O}$-notation is like the O-notation except that logarithm factors in n and in L are ignored. In the subdivision setting, "near-optimality" may be taken to be $\widetilde{O}(n^2(n+L))$.

Our central problem is this: given a box B_0, to recursively split B_0 into subboxes until each subbox B satisfies $C_0(B) \vee C_1(B)$. The recursive splitting forms a tree $\mathcal{T}(B_0)$ of boxes with B_0 at the root, with each internal node B failing $C_0(B) \vee C_1(B)$. We assume some scheme for splitting a box B into a subset $B_1, \ldots, B_k$ where $\{B_1, \ldots, B_k\}$ is a subdivision of B. A simple scheme is to let $k = 2^m$ and the B_i's are congruent to each other. There are also various binary schemes where $k = 2$. In the binary schemes, it is necessary to ensure that the aspect ratios of the subboxes remain bounded. Assuming that k is a constant in the splits, each internal node in $\mathcal{T}(B_0)$ has degree k. If $\mathcal{T}(B_0)$ is finite, then the leaves of $\mathcal{T}(B_0)$ form a subdivision of B_0. We are mainly interested in the subdivision $\mathcal{S}(B_0)$, comprising those leaves that fail $C_0(B)$ (thus satisfying $C_1(B)$).

Let Q_0, Q_1 be queues of boxes, with the usual queue operations (push and pop) for adding and removing boxes. Consider the following subroutine to compute $\mathcal{S}(Q_1)$. We may, for instance, initialize Q_1 to $\{B_0\}$.

```
Subdivide Routine
   INPUT: Q_1
   OUTPUT: Q_2
      Q_2 ← ∅
      While Q_1 is non-empty
         B ← Q_1.pop()
         If C_0(B) fails
             If C_1(B) holds,
                Q_2.push(B) //i.e., output B
             Else
                Q_1.push(split(B))
```

The main correctness question about the `Subdivide` routine is termination: does Q_1 eventually become empty? This is equivalent to every box B eventually satisfying $C_0(B) \vee C_1(B)$ (if a box is split, this consideration is transferred to its children). For instance, in real root isolation, B_0 is an interval and we have termination iff there are no multiple roots in B_0. For path planning, we modify $C_0(B)$ to $C_0(B) \vee C_\varepsilon(B)$ where $C_\varepsilon(B)$ holds if the width of B is less than ε. Therefore, to ensure termination, we must either restrict the input (e.g., there are no multiple roots), or introduce suitable ε-correctness concepts (such as resolution-exactness in path planning in Section 2(b)).

We view `Subdivide Routine` as the centerpiece of our algorithm. Its output is the queue Q_2 containing the subdivision $\mathcal{S}(B_0)$. For instance, the EVAL and CEVAL algorithms in [42] are basically this subroutine. But in general, we expect to do some post processing of Q_2 to obtain the final result. For example, we may have to construct the simplicial complex K representing the zero set $\texttt{Zero}(\boldsymbol{f})$ [39,31,32]. Likewise, we may need to do some initialization to prepare for subdivision. For example, in path planning, we need to first ensure that the start α and goal β configurations are free [50]. This suggests that we need an initialization phase before the `Subdivide Routine`, and we need a construction phase after. Following [31], we may assume that input and output for each phase are appropriate queues. We are ready to present a simple form of this framework:

```
Simple Subdivision Framework
   INPUT: B_0, ε, ...
   OUTPUT: Q_3
      I. Initialization Phase
            Q_1 ← Preprocessing(B_0)
      II. Subdivision Phase
            Q_2 ← Subdivide(Q_1)
      III. Construction Phase
            Q_3 ← Construct(Q_2)
```

We can derive algorithms for the illustrative problems (A) and (B) using this framework. This amounts to instantiating the routines in the three phases. A key idea in our design of these routines is to make the subdivision phase do most of the work, i.e., its complexity ought to dominate that of the other two phases. This is not true for all subdivision algorithms: an example is Snyder's approach to isotopic curves and surfaces [47] (see [9, Chap.5.2.3]). The plausibility of our key idea comes from the fact that when subdivision is fine enough, everything would be "as simple as possible", modulo singularities. Singularities, even isolated ones, can be arbitrarily complex. For example, the neighborhood of a degenerate Voronoi vertex can have arbitrarily high degree. We may simply exclude singularities by fiat (as in isotopic curves [39,31] or in arrangement of curves [30]). But our ultimate goal is not to avoid singularities but to introduce soft-ε notions (as in root isolation [4] or in Voronoi diagrams [6]). We design the $C_0(B)$ and $C_1(B)$ to capture the non-degenerate situations outside of such singularities. We say that output Q_2 of `Subdivide(`Q_1`)` is "fine enough" if *the cost of constructing the final output is* $O(1)$ *per box in* Q_2. In the problems of isotopic curves and surfaces [39,31,32], the output is a planar embedded graph (for curves) or a triangulation (for surfaces). When the subdivision is "fine enough", we only need to construct simple, almost-trivial, graphs or triangulations $G(B)$ in $O(1)$ time for each $B \in \mathcal{S}(B_0)$. The output is the union of these $G(B)$'s. Thus, the global complexity of these algorithms is indeed dominated by the subdivision process. This key idea ensures that the resulting algorithm is easy implementable or practical. A caveat is that the complexity may become a bottle neck in higher dimensions. Nevertheless, it ensures that we could solve such problems, at least in small regions-of-interest.

How good is the proposed framework? For real root isolation of integer polynomials, the size of the subdivision tree $\mathcal{T}(B_0)$ is near-optimal [45,12]; the analysis can be greatly generalized [14], including accounting for bit complexity. The complexity of the PV algorithm in higher dimensions has also been analyzed [15,16]. For top performance in univariate complex root isolation [5,4] it is necessary to introduce Newton iteration and to maintain more complicated data structures ("components") in order to achieve near-optimal bounds. See [52, §1.1] for a subdivision framework that incorporates Newton iteration. Newton iteration will produce non-aligned boxes, i.e., boxes that do not come from repeated splits of B_0. This is not an issue for root isolation but in geometric problems such as arrangement of curves [30] and Voronoi diagrams [6], non-aligned boxes (called root boxes) arise where it was necessary to provide "plumbing" so that the non-aligned boxes "conforms" with the rest of the aligned boxes.

We generally need to maintain adjacency relations among boxes in $\mathcal{S}(B_0)$, especially for the construction phase. Two boxes B, B' are **adjacent** if $B \cap B'$ has codimension 1. There is a general technique to efficiently maintain such information, namely to ensure that the subdivision $\mathcal{S}(B_0)$ is **smooth** [7]. Smoothness means that if $B, B' \in \mathcal{S}(B_0)$ are two adjacent boxes, then their depths in the tree $\mathcal{T}(B_0)$ differ by at most 1. This can be done systematically by (1) maintaining "principal neighbor" pointers for each box and (2) perform `smoothSplit(`B`)` instead of `split(`B`)` in `Subdivide(`Q_1`)`. In `smoothSplit(`B`)`, we split B and recursively split any adjacent boxes necessary to maintain smoothness. Although a single `smoothSplit(`B`)` can be linear in the size of Q_1, we show in [7] that this operation has amortized $O(1)$ complexity, and hence does not change the overall complexity.

So far, we have assumed subdivision in $\mathbb{R}^n$. What about subdivision in non-Euclidean spaces? Burr [14] has provided an account of subdivision in abstract measure space, aimed at amortized complexity analysis. We take a different approach, with an eye towards implementation rather than analysis: in path planning, we need to perform subdivision in configuration spaces $\mathcal{C}space$. Such spaces are typically non-Euclidean: $\mathcal{C}space = \mathbb{R}^2 \times \mathbb{T}$ where $\mathbb{T}$ is the torus [56], $\mathcal{C}space = \mathbb{R}^3 \times S^2$ [23], and $\mathcal{C}space = SE(3) = \mathbb{R}^3 \times SO(3)$ [55]. Using the analogy of charts and atlases in manifold theory, we define charts and atlases for subdivision. Furthermore, we generalize boxes to general shapes called "test cells" that include simplices or convex polytopes which have bounded aspect ratios. Resolution-exact planners (Section 2(b)) can be achieved in such settings and with an accuracy constant given by $K = C_0 D_0 L_0 (1 + \sigma)$ where C_0, D_0, L_0, σ are constants associated with (respectively) the atlas, subdivision scheme, a Lipshitz constant and effectivity factor [55]. It is also clear that we could extend subdivision atlases to projective spaces ($\mathbb{RP}^n$ and $\mathbb{CP}^n$).

In our abstract, we said that algorithms in the subdivision framework are "effective" in the sense of *easily and correctly implementable from standard algorithmic components*. The preceding outline exposes some of these algorithmic components: queues, subdivision structures, boxes with adjacency links, union find data structure, etc. But the critical issue of numerical approximation is deferred to the next section.

4 Numerical Precision in Subdivision Framework

The main problem of subdivision is when to stop, and this is controlled by predicates. In our `Subdivide` Subroutine, we used two logical box predicates $C_0(B)$ and $C_1(B)$. Both are typically reduced to some form of sign computation: in the PV algorithm [39], $C_0(B)$ is defined as $[[0 \notin f(B)]]$ for some continuous function $f : \mathbb{R}^n \to \mathbb{R}$. As for $C_1(B)$, we follow a nice device of [15] for describing this predicate: first define

$$\nabla^{(2)} f : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$$

where $\nabla^{(2)} f(\boldsymbol{x}, \boldsymbol{y}) = \langle \nabla f(\boldsymbol{x}), \nabla f(\boldsymbol{y}) \rangle$ and $\langle \cdot, \cdot \rangle$ denotes the dot product. For instance, for $n = 2$, $\nabla^{(2)} f(\boldsymbol{x}, \boldsymbol{y}) = \partial_1 f(\boldsymbol{x}) \cdot \partial_1 f(\boldsymbol{y}) + \partial_2 f(\boldsymbol{x}) \cdot \partial_2 f(\boldsymbol{y})$ where ∂_i denotes partial derivative with respect to x_i. Then $C_1(B)$ is $[[0 \notin \nabla^{(2)} f(B, B)]]$. Both $C_0(B)$ and $C_1(B)$ are mathematically exact formulations, but far from effective.

We now sketch a 3 stage development to systematically derive an implementable form, following [52]. The outline may be illustrated by using the $C_0(B)$ predicate: we first define an interval version of $C_0(B)$ denoted $\Box C_0(B)$. Then we modify the interval version to an "effective" version denoted $\widetilde{\Box} C_0(B)$. These version are connected through a chain of logical implications:

$$\widetilde{\Box} C_0(B) \Rightarrow \Box C_0(B) \Rightarrow C_0(B).$$

Each of these predicates are, in turn, based on underlying functions on boxes: $\Box C_0(B)$ is $[[0 \notin \Box f(B)]]$ and $\widetilde{\Box} C_0(B)$ is $[[0 \notin \widetilde{\Box} f(B)]]$. We must now define the functions $\Box f$ and $\widetilde{\Box} f$ for any f.

Our numerical algorithms are intended to be certified in the sense of interval arithmetic [34,26]. But we wish to carry our subdivision algorithms in a slightly more general setting, say in normed vector spaces X, Y. Here, we can do differentiation (as in $\nabla^{(2)} f$) and do dilation of boxes or balls (as in $B \mapsto 2B$). Suppose we have a function $f : X \to Y$. Define the **natural set extension** of f to be

$$f : 2^X \to 2^Y \tag{4}$$

where $f(S) := \{f(x) : x \in S\}$ for $S \in 2^X$. We are[5] "overloading" the symbol f in (4). But if we identify the elements of X with the singletons in 2^X, we see that this extension is natural, and justifies reuse of the symbol f. Again, 2^X is too big and we restrict f to the nice subsets of X. Let $\Box X$ and $\Box Y$ be the collection of nice subsets of X and Y. Note that even if $B \subseteq X$ is a nice set, $f(B)$ need not be nice (except when $Y = \mathbb{R}$). In other words, the function (4) does not naturally induce a function of the form

$$F : \Box X \to \Box Y. \tag{5}$$

Thus we are obliged to explicitly define the function F in (5). What is the relation between f and F? We call F a **box form** of f provided it is **conservative relative to** f (i.e., $f(B) \subseteq F(B)$) and **convergent to** f (i.e., if $\{B_i : i \geq 0\}$ converges to a point $p \in X$, then $\lim_{i \to 0} F(B_i) = f(p)$). We may write "$F \to f$" if F is convergent to a point function f. This parallels our definition of soft predicates. We write "$\Box f$" for a generic box form of f. If it is necessary to distinguish different box forms, we use subscripts such as $\Box_2 f$. The function (5) is called a **box function** when it is the box form of some f. The interval literature defines many box forms for $f : \mathbb{R}^n \to \mathbb{R}$. For example, the **mean value form** of f given by

$$\Box_{\text{M}} f(B) := f(m(B)) + \Box \nabla f(B)^T \cdot (B - m(B)) \tag{6}$$

where $m(B)$ is the midpoint of B and $\nabla f = (f_1, \ldots, f_n)^T$ is the gradient of f, with $f_i = \partial_i f$. Our definition of mean value form invokes another box form $\Box \nabla f(B) = (\Box f_1, \ldots, \Box f_n)^T$. Since this second box form is generic, $\Box_{\text{M}}$ is still not fully unspecified.

Suppose F is a box form of f. By regarding X as a subset of $\Box X$, we can view f as the restriction of F to X, i.e., $f = F|_X$. Let $F_i : \Box X \to \Box Y$ $(i = 1, 2)$ be two functions (not necessarily box forms). Write $F_1 \subseteq F_2$ if for all $B \in \Box X$, $F_1(B) \subseteq F_2(B)$. Then we have

Let $F_1 \subseteq F_2$. If F_2 is a box form, then F_1 is a box form.

[5] Some authors introduce a new symbol, say F, to signal this change.

Of course, we also have $F_1|_X = F_2|_X$. What we need in our application, however, is the "converse": *if* $F_1 \subseteq F_2$ *and* F_1 *is a box form, then* F_2 *is a box form.* To motivate this application, consider the mean value form $\square_{\text{M}}$: its definition (6) calls for an exact evaluation $f(m(B))$, which we must approximate. In general, for any interval form $\square f$, we need to approximate it by some function of the type $\widetilde{\square} f : \square\mathbb{R}^m \to \square\mathbb{R}$. But how are $\widetilde{\square} f$ and $\square f$ related? We will say $\widetilde{\square} f$ an **effective** form of $\square f$ provided these properties hold:

(i) (Inclusion) $\square f \subseteq \widetilde{\square} f$ and
(ii) (Precision) $q(\square f(B), \widetilde{\square} f(B)) \leq w(B)$ where $w(B)$ is the width of B and $q(I, J)$ is the Hausdorff metric on closed intervals.
(iii) (Exactness) $\square f$ is dyadically exact.

We will discuss the third property (iii) below. But first, we note that properties (i) and (ii) ensure our desired converse:

Lemma 1. *If* $\widetilde{\square} f$ *satisfies (i) and (ii), then* $\widetilde{\square} f$ *is a box form of* f*.*

To compute $\widetilde{\square} f(B)$, this lemma says that, provided our numerical approximation is rounded correctly to satisfy Property (i), then we only have to ensure that the error is bounded by the width of B as in Property (ii). Although the boxes B are distributed over space and time, the global correctness is guaranteed by the nature of our predicates.

We now turn to Property (iii). This requirement is connected to general ideas about efficiency and effectivity of numerical computation. For this, we assume that $X = \mathbb{R}^n$ and $Y = \mathbb{R}$. In practice, real numbers are most efficiently approximated by dyadic numbers, $\mathbb{Z}[\frac{1}{2}]$ or BigFloats (see [57]). Our definition of $\widetilde{\square} f$ serves the fiction that it could accept every box in $\square\mathbb{R}^n$. This is useful fiction because it cleanly fits into mathematical analysis. But in implementations, these box functions only need to accept **dyadic boxes**, i.e., boxes whose corners have dyadic coordinates. We say a box function $F : \square\mathbb{R}^n \to \square\mathbb{R}$ is **dyadically exact** if its restriction to dyadic boxes outputs dyadic intervals. This explains our Property (iii). Evidently, it is not hard satisfy all 3 properties of effectivity.

Literate[6] **Algorithmic Development.** In [52], we developed a subdivision algorithm for isolating the simple real roots of a real system

$$\boldsymbol{f} = (f_1, \ldots, f_n) : \mathbb{R}^n \to \mathbb{R}^n.$$

As a subdivision algorithm, it has several predicates: the centerpiece is the Miranda Test $\texttt{MK}(B)$ for existence of real roots in B. We have our ubiquitous exclusion test, but defined as $C_0(B) := [[(\exists i = 1, \ldots, n)(0 \notin f_i(B))]]$. We also need a Jacobian Test $\texttt{JC}(B)$ to confirm at most one root. Each predicate C is first defined mathematically, then as box predicates $\square C$, and finally as effective predicates $\widetilde{\square} C$. Thus, there are three levels of description:

(A) **Abstract** C, f
(I) **Interval** $\square C$, $\square f$
(E) **Effective** $\widetilde{\square} C$, $\widetilde{\square} f$

As expressed by Burr et al. [15, §2.3], the goal is to delay the introduction of $\square C$ (and hence $\widetilde{\square} C$). The motivation comes from the fact that the theory is cleanest at Level (A), and less so at later levels. In effect, we have three algorithms:

$$A, \quad \square A, \quad \widetilde{\square} A \tag{7}$$

each being an instantiation of a common algorithmic scheme by predicates and functions of the appropriate level. This is analogous to standard construction of numerical algorithms from $A \mapsto \widetilde{A}$ (see the Introduction); the difference is that our starting point A is in the Subdivision Framework. We then prove the algorithms correct at each level. At each level, we bring in new details but are able to rely on the properties already proved in the previous level. For instance, an important phenomenon when we transition from A to $\square A$ is the appearance of Lipshitz constants inherent in interval methods. This approach ("AIE methodology") displays a continuity of ideas and exposes the issues unique to each level. The clarity and confidence in the correctness of $\widetilde{\square} A$ are surely much better than if we had attempted[7] an *ab initio* correctness proof of $\widetilde{\square} A$. Quoting Knuth:

"Beware of bugs in the above code; I have only proved it correct, not tried it."

[6] In the spirit of Knuth's "Literate Programming".
[7] It is possible that such proofs contribute to the poor reputation of error analysis as a topic.

5 On Oracle Objects

In our `Simple Subdivision Framework`, we pass queues from one phase to the next. Such queues serve to represent intermediate states of our ultimate output (the simplex K for Problem A or the path π for Problem B). In this section, we explore the idea of representing computational objects that encode states and other information. The term "object" suggests connection to Object Oriented Programming Languages (OOPL) since, in order to make our algorithms effective, it must be ultimately implemented in a programming language. See the recent paper of Braußе et al. [10] that also brings programming semantics into the theory of real computation.

5.1 Soft Tests

We begin by discussing the "ur-predicate", the comparison of two real numbers x, y. We may write the comparison as a logical predicate $[[x < y]]$, using the test notation of Section 1.1. If true, we branch to point A, and otherwise we branch to point B. In exact computation, the two points A and B in the program encode the respective assertions $[x < y]$ and $[x \geq y]$, where the notation "$[S]$" is now an **assertion** that S is true. In numerical computation, we might need point C in the program to encode the assertion $[x \ ? \ y]$ (don't know). To simplify our primitives, let us reduce this 3-valued test to a 2-valued version, denoted $[(x < y)]$ where point A represents the assertion $[x < y]$ (as before) but point B asserts $[x \geq ?y]$. But outcome "$[x \geq ?y]$" suggests the assertion $[x \geq y] \vee [x \ ? \ y]$ (we will explore this more carefully below). In reality, we reached the point B because the test $[(x < y)]$ was done with limited precision p. Thus we may explicitly indicate this precision[8] by writing $[(x < y)]_p$. We call $[(x < y)]_p$ a **one-way test** because the failure to assert $[x < y]$ does not imply the negated assertion $[x \geq y]$.

The 2-valued exact tests $[[x < y]]$ and $[[x \geq y]]$ are equivalent in the sense that one is obtained from the other by switching truth values. But $[(x < y)]_p$ and $[(x \geq y)]_p$ have no such symmetry. This suggests that we could define another form of $[(x < y)]_p$ in which the point B encodes the assertion $[x \geq y]$, but point A encodes $[x <?y]$. These two versions of the one-way test have their respective uses – the first version is aimed at confirming the assertion "$[x < y]$", and the second version is aimed at falsifying it. To distinguish them, let us write $Con[x < y]_p$ for confirmation test and $Fal[x < y]_p$ for other. Unless otherwise stated, we continue to view the test $[(x < y)]$ in the confirmation mode. It might appear that we are splitting hairs by reducing a 3-way test to two 2-way tests. But since these tests may involve heavy computations (such as testing if a robot configuration is free), this split may be useful. Alternatively, the numbers x, y may represent complicated expressions (see below).

To implement such one-way tests, we need to assume that x, y are **number oracles** (see [29]). That means for each $p \in \mathbb{Z}$, we can ask for a p-bit approximation of x, denoted $(x)_p$. This[9] means $(x)_p = x \pm 2^{-p}$. We may represent $(x)_p$ by a dyadic number with at most p bits after the binary point. For instance, we can implement the one-way $[(x < y)]_p$ as follows:

```
if (x)_p + 2^-p < (y)_p - 2^-p
    return [x < y]
else
    return [x ≥?y]
```

Observe that this algorithm is indeterminate (see Section 2) because $(x)_p$ does not identify a specific value, but depends on the oracle for x. The exact test $[[x < y]]$ can be reduced to two one-way tests as follows:

[8] We use "precision" for the *a priori* user-specified bound. The algorithm delivers a value whose *a posteriori* error is at most this precision.

[9] We write $a = b \pm c$ to mean there exists a constant $\theta \in [-1, 1]$ such that $a = b + \theta \cdot c$. Alternatively, $|a - b| \leq |c|$.

```
Subroutine [[x < y]]:
   For p = 0, 1, ...
       if Con[x < y]_p
          return [x < y]
       else if Fal[x < y]_p
          return [x ≥ y]
```

In the case $x = y$, this subroutine is non-halting. Unfortunately, this is the best we can do without more information about x or y. It turns out that we can modify the loop above to produce a halting subroutine. That is the **Soft Zero Test** in [54] which has three outcomes: $[x < y]$, $[x > y]$ and $[x \simeq y]$. The last outcome is new, and is defined[10] to mean

$$\left[\frac{1}{2}x < y < 2x\right] \quad \vee \quad \left[\frac{1}{2}x > y > 2x\right]. \tag{8}$$

The first disjunct implies that x, y are both positive, and the second implies both are negative. We denote this test by $[\{x : y\}]$. What makes this test decidable (halting) is the introduction of the new outcome. But we also need a "mild" assumption: *either x or y is non-zero*. It is assumed that both x and y are non-negative in [54]. That is justified by the intended application where both x and y are sums of absolute values (from the Pellet Test). Essentially, this Soft Zero Test is at the heart of our soft-ε criteria for roots. In exact computation, comparing two numbers $x : y$ is equivalent to the computing the sign of the single number $x - y$. The Soft Zero Test shows that you can do a bit more by keeping x and y separate.

What is the logical status of the intuitive formula "$[x >? y] = [x > y] \vee [x? y]$"? The truth-values $[x < y]_p$ are parametrized by x, y and also p. It is enough to consider the non-parametric setting where, in addition to `true`, `false`, we add a third logical value, `frue` (false-or-true). Then we have these truth tables:

$\wedge$	true	frue	false
true	true	frue	false
frue	frue	frue	false
false	false	false	false

	true	frue	false
$\neg$	false	frue	true

$\vee$	true	frue	false
true	true	true	true
frue	true	frue	frue
false	true	frue	false

So far, we looked at point-based comparisons. We now consider interval-based comparisons. The ur-predicate here is the **Membership Test** $[[x \in I]]$ where I is an interval. Here, we view I as a dyadic interval and x is the usual oracle. Let $[\{x \in I\}]$ denote the **Soft Membership Test** with two outcomes, $[x \mathrel{\widetilde{\in}} I]$ and $[x \mathrel{\widetilde{\notin}} I]$. We define them as $[x \in 2I]$ and $[x \notin I/2]$, respectively. If desired, we may replace '2' by $1 + 2^{-p}$ and denote it by $[\{x \in I\}]_p$. It is indeterminate because, in case $x \in 2I \setminus I/2$, both outcomes are acceptable. This can be implemented as the exact test

$$[[(x)_p \in I]] \tag{9}$$

since $p = -\lceil 1 + \log_2(w(I))\rceil$ is easily computable. Exactness of (9) follows from the fact that I and $(x)_p$ are dyadic. If $[(x)_p \in I]$, we output $[x \mathrel{\widetilde{\in}} I]$, and otherwise, we output $[x \mathrel{\widetilde{\notin}} I]$. The Soft Membership Test is unconditional. These ideas can be generalized to produced soft membership in balls or boxes: the predicates in [42,5] are examples of such tests.

5.2 Whence Number Oracles?

Number oracles are ubiquitous in the theory of real computation [51,29]. Their availability is largely assumed. Perhaps it is generally assumed that they come from well-known mathematical series, and all we need to do is to evaluate such series to enough precision. But even this problem deserves careful investigation from the viewpoint of complexity. For instance, the family of hypergeometric functions provide us with a rich class of series that include the elementary functions and much more. But if we are given a function in terms of its hypergeometric parameters, there are issues of transforming them to speed up the convergence. From the work of Richard Brent in the 1970s, it is well-known that to evaluate such functions at a fixed point is extremely fast (basically the speed of integer multiplication, perhaps with extra log factors). But this tells us little about global or uniform complexity of these approximation algorithms. We refer the interested reader

[10] Despite the appearance of asymmetry, x and y are treated symmetrically by this definition.

to [18,19]. In algebraic computation, we do not have such series. But it is easy to provide an oracle for any algebraic number α if we have a defining integer polynomial, $p(\alpha) = 0$ with $p'(\alpha) \neq 0$. If α is real, we can find an isolating interval $[a, b]$ for α. Thereafter, we can use bisection to produce a convergent sequence of intervals. Following Dekker and Brent [11], we can use a Newton-bisection iteration to speed up this process. A recent variant of Newton-bisection from Abbot, Sagraloff and Kerber [1,27] led to the complexity analysis of such speedups. There are analogous procedures to produce oracles for complex α. It turns out that in geometric computations, we seldom begin with algebraic numbers: instead we typically start with rational numbers and α is built up as an expression using different algebraic operators to produce arbitrary algebraic numbers. Let us briefly describe this class of oracles.

Let Ω be a set of real algebraic operators. Typically, Ω contains at least $\{\pm, \times, \div, \sqrt{\cdot}\} \cup \mathbb{Z}$. Assume that each operator $\omega \in \Omega$ has an approximation algorithm [57]. Let $E(\boldsymbol{x}) = E(x_1, \ldots, x_n)$ be an algebraic expression over $\{x_1, \ldots, x_n\} \cup \Omega$. E.g., $E(\boldsymbol{x}) = \sqrt{x^2 - 2y + 1} - \sqrt[3]{xy - y^2}$. If $\boldsymbol{a} = (a_1, \ldots, a_n)$ is a sequence of number oracles, then there are general mechanisms to construct an oracle for the number $E(\boldsymbol{a})$ (see [33,59]). Note that this description is more general than the usual setting for EGC where the expression E is a constant; but the extension to expressions with arguments is relatively direct straightforward. Zero Problems arise from the fact that some operators such as $\div, \sqrt{\cdot}$ or log are partial functions, and so $E(\boldsymbol{a})$ may be undefined. We need to detect this situation and halt. This is a hopeless case, even for $E(\boldsymbol{a}) = a_1 - a_2$, unless we have prior information such as a_1, a_2 are algebraic with degree and height bounds (or some height substitute).

5.3 Cluster Oracles

Oracles arising from algebraic expressions can be generalized to geometric expressions in the sense of Constructive Solid Geometry (CSG) [2]. Here, the expressions are built from primitive geometric shapes such as numbers, points, balls and half-spaces, using boolean operators such as intersection and union. Such expressions can be new sources for number oracles. But in this subsection, we will focus on a recent extension of number oracles to "cluster oracles". It arose in our root clustering algorithms [5,4], and its extension to solving triangular systems in $\mathbb{C}^n$ [25]. Intuitively, Cauchy sequences must be generalized to "Cauchy trees" because clusters may split upon request for more precision.

Multisets arise naturally when we consider the zero sets of functions: let $D \subseteq \mathbb{C}^n$ and $\boldsymbol{f} : \mathbb{C}^n \to \mathbb{C}^n$. Assume that $\boldsymbol{f}^{-1}(\boldsymbol{0}) \cap D$ is a finite set, and for each $\boldsymbol{a} \in \boldsymbol{f}^{-1}(\boldsymbol{0})$, we can assign an integer $\mu(\boldsymbol{a}) \geq 1$ called its multiplicity. We introduce two useful notations: let $\texttt{Zero}_{\boldsymbol{f}}(D) := D \cap \boldsymbol{f}^{-1}(\boldsymbol{0})$ and $\#_{\boldsymbol{f}}(D)$ be the total multiplicities of the roots in $\texttt{Zero}_{\boldsymbol{f}}(D)$. The pair $(\texttt{Zero}_F(D), \#_F(D))$ is an example of a multiset.

In general, a **multiset** S is a pair $(\underline{S}, \mu)$ where $\underline{S}$ is[11] an ordinary set (called the **underlying set** of S) and $\mu = \mu_S$ assigns a **multiplicity**, a positive integer $\mu(x)$, to each $x \in \underline{S}$. Also let $\mu(S) := \sum_{x \in \underline{S}} \mu(x)$ be the (total) **multiplicity** of S, assumed to be finite. Let $|S|$ denote the cardinality of $\underline{S}$; so $|S| \leq \mu(S)$. If $|S| = 1$ (resp., $|S| = 0$) we say S is a **singleton** (resp., **empty**). We denote the empty multiset as well (ordinary) empty set by the same symbol $\emptyset$. If T is another multiset, we write $S \subseteq T$ and call S a **subset** of T if $\underline{S} \subseteq \underline{T}$ and $\mu_S(x) \leq \mu_T(x)$ for all $x \in \underline{S}$. In this paper, we assume[12] equality, i.e., $\mu_S(x) = \mu_T(x)$, in subset relations. The intersection $S \cap T$ is the largest multiset R such that $R \subseteq S$ and $R \subseteq T$.

Our multisets interact with the world of ordinary sets: let X be an ordinary set. Then '$S \subseteq X$' means that $\underline{S} \subseteq X$. Likewise '$S \cap X$' denotes the multiset $T \subseteq S$ where $x \in \underline{T}$ iff $x \in X$.

We are interested in the concept of a "cluster" C. Informally, a cluster C is a multiset in a larger multiset U which is nicely separated from $U \setminus C$. Let us formulate this concept in the context of a normed linear space X with norm $\|\cdot\|$. Let $\Delta = \Delta(m, r) \subseteq X$ denote the **ball** centered at m of radius $r \geq 0$. For real $\alpha > 0$, let $\alpha\Delta(m, r)$ denote the ball $\Delta(m, \alpha r)$. Let us fix a multiset $U \subseteq X$. A **cluster** (of U) is a non-empty set $C \subseteq U$ of the form $C = U \cap \Delta$ for some ball $\Delta \subseteq X$. Call such a Δ an **isolator** of cluster C; this isolator is **natural** if, in addition, $C = U \cap 3\Delta$. If C has a natural isolator, we call it a **natural cluster**. The fundamental property of natural clusters is this:

[11] There should no confusion with the notion support of a simplicial complex $\underline{K}$.

[12] Strict inequality may arise in subsets of zero sets: if $F = (F_1, \ldots, F_n)$ where F_i are polynomials, and $G = (G_1, \ldots, G_n)$ where each G_i divides F_i, then $\texttt{Zero}_G(D) \subseteq \texttt{Zero}_F(D)$ might exhibit this phenomenon.

Lemma 2. *Let X be a normed linear space and $U \subseteq X$ be a finite multiset. Then any two natural clusters C, C' of U are either disjoint or have a subset relation, i.e., either $C \cap C' = \emptyset$ or $C \subseteq C'$ or $C' \subseteq C$.*

Basically the proof works because of the triangle inequality in X. As a corollary, there are at most $2|U| - 1$ natural clusters of U, and they can be organized into a **cluster tree**: each node in the tree is a natural cluster of U and the child-parent relation is just $C \subset C'$. The original cluster concept in [53,4] assumes $X = \mathbb{C}$. In [25] it was extended to $X = \mathbb{C}^n$, for complex roots of triangular systems. Computing natural clusters may be regarded as the soft-ε criterion for root isolation; as we shall see, it is effective and can completely remove the Zero Problems associated with multiple roots.

But how do we compute such clusters? We need predicates to check if a given Δ is an isolator and to determine its total multiplicity. For $X = \mathbb{C}^n$, we could use some multidimensional form of Pellet's test, and for $X = \mathbb{R}^n$, there are similar tools such as multidimensional Sturm theory based on quadratic forms [38,36,37]. Unfortunately, at present, these tools do not appear to be practical. In lieu of this, we take another route in [25]: we first reduce the input system into triangular systems using known algebraic techniques. In the triangular form, we can compute the multivariate clusters and their multiplicities using the efficient univariate multiplicity tests of [4,24].

The main tool in [4,24] is a test from Pellet (1881). Fix a complex polynomial $f(z) \in \mathbb{C}[z]$. First consider the test

$$T_k(f) := \left[\!\left[\; |a_k| > \sum_{i \neq k} |a_i| \; \right]\!\right] \tag{10}$$

where $f(z) = \sum_{i=0}^{n} a_i z^i$. This test is defined for any $k = 0, \ldots, n$. Pellet's theorem says that

$$\textit{if } T_k(f) \textit{ succeeds then } \#_f(\Delta(0,1)) = k.$$

So this test, which is a simple application of Rouché's Theorem, can confirm that the total multiplicity of the complex roots of f in the unit disc $\Delta(\mathbf{0}, 1)$ centered at the origin is exactly k. The case $k = 0$ is interesting – it is an exclusion test! It is more expensive than the standard C_0 test, but we shall see that its failure provides some partial converse information. We can confirm that the disc $\Delta(0, r)$ of radius $r > 0$ has k roots by applying the T_k-test to the polynomial $f(rz) = \sum_i b_i z^i$ where $b_i = a_i r^i$. Similarly, we can confirm that $\Delta(m, 1)$ (the unit disc centered at $m \in \mathbb{C}$) has k roots by applying the T_k-test to the Taylor-shifted polynomial $f(z + m)$. Combining these two operations, we obtain a test for an arbitrary disk $\Delta(m, r)$. Let $T_k(f, m, r)$, or simply T_k, denote such a test.

The next question is crucial: *when is the success of T_k test assured?* It is shown that there are positive numbers $c < 1 < C$ such that if

$$\#_f(c \cdot \Delta(m, r)) = \#_f(C \cdot \Delta(m, r))$$

then $T_k(f, m, r)$ will succeed. In other words, this gives a partial converse to Pellet's test. Unfortunately, these numbers depend on the degree: $c = \Omega(1/n)$ and $C = O(n^3)$. By applying Graeffe iteration $5 + \log\log n$ times to f, we reduce these numbers to $c = 2\sqrt{2}/3 \simeq 0.94$ and $C = 4/3$. Suppose B is a subdivision box and $\Delta(B)$ is its circumscribing disc. Let $T_k^G(f; B)$ denote the application of the T_k test to the Graeffe-transformed Δ-shifted polynomial f. Choose $k = 0$:

$$\begin{array}{l} \textit{If } T_0^G(f; B) \textit{ succeeds, } \#_f(B) = 0. \\ \textit{If } T_0^G(f; B) \textit{ fails, } \#_f(2B) > 0. \end{array}$$

We classify boxes as **excluded** if this test succeeds, and **included** otherwise. We now have a very powerful test that is analogous to the Soft Membership Test earlier. The remaining issues treated in the paper are:

- We need approximate versions of these tests: thus we use $\widetilde{T}_k^G$ instead of T_k^G. The Soft Zero Test is used to make numerical comparison of (10).
- We use $\widetilde{T}_0^G(B)$ as exclusion test. We maintain the connected components of those B's that are included (i.e., fail the exclusion test). These components are potentially cluster. We refine a component by splitting each of its constituent boxes, and applying the $\widetilde{T}_0^G$ tests again.
- To obtain near-optimal bounds, we use the Abbot form of Newton-Bisection [1,27] on a connected component C. If C is sufficiently separated from the other components, then we could use the $\widetilde{T}_k^G$ test to determine that $\#(C) = k$, and even apply the order k Newton iteration successfully.

[14] For complexity analysis (in the bench mark case), we need charging schemes that charge these tests to roots of f in $2B_0$. It turns out that for the non-integer polynomials, we can provide some complexity estimates based on the root geometry.

We hope this overview may make the original papers more accessible. In [25], we package the above structures into **cluster oracles** in order to compute multi-dimensional clusters inductively. Such cluster oracles, viewed as objects in the sense of OOPL, can provide an efficient mechanism for other similar applications.

6 Conclusion and Open Problems

The foundations of subdivision computation is a wide-open area of research, with promises of new and effective algorithms that have mild (or no) conditions on the input. Our illustrative examples suggest that such algorithms can be practical and compare favorably with less-rigorous solutions or symbolic or exact solutions.

Our soft-ε criteria for two key problems seems to have achieved a satisfactory level of completeness: (a) complex root clustering for polynomials with oracle coefficients [4] and (b) resolution-exact path planning [50]. Of course, success creates its own (new) set of problems: for (a), we would like to treat more general functions such as analytic or harmonic functions. For (b), the challenge is to design a practical nonheuristic planner for spatial robots with 6 degrees-of-freedom. This natural but elusive quest appears to be reachable within our framework. Finally, we pose some open problems:

1. Algorithms with soft-ε correctness is the continuous analogue of "approximation schemes" in discrete optimization algorithms. Just as the barrier to polynomial-time schemes (PTAS) are located in NP-hardness or similar complexity classes, the barrier in the continuous case are various Zero Problems. We would like a complexity theory of such Zero Problems. See also the recent paper [17].
2. It is generally challenging to remove *all* Zero Problems. A prime example is the PV-type algorithms [39]. Such algorithms are based on the Marching Cube paradigm, and require the exact sign evaluation of a function at the corners of subdivision boxes. How do we soften this?
3. Interval methods are central to all our algorithms. We would like to develop interval methods in more abstract spaces than Euclidean ones. Are normed vector spaces or metric spaces the natural home for such extensions? As usual, we need good problems on which to cut our teeth.
4. Path planning in very high dimensions is an open problem. An example of a currently out-of-reach path planning problem: a planar snake with 10 joints. The configuration space is $\mathbb{R}^2 \times (S^1)^{10}$. This requires new paradigms, but we believe they can be built upon a subdivision framework.
5. Path planning is only the simplest of motion planning problems. What do soft-ε algorithms mean in non-holonomic planning, or kino-dynamic planning? A good problem is to try subdivision in state space: imagine a point robot in the plane amidst obstacles. Its state or coordinates are $(x, y, \dot{x}, \dot{y})$ representing position and velocity. We want to plan a minimum time trajectory from some start to goal states, subject to acceleration bounds.
6. The notion of natural root clusters suggests other applications and extensions. How do we cluster matrix eigenvalues? It seems that other considerations come into play: the invariant subspaces associated with eigenvalues should play a role in defining "natural clusters of eigenvalues".
7. Complexity analysis of subdivision is largely open. The case of univariate zeros is reasonably well-understood, but there are many open problems even for zero-dimensional problems in higher dimensions. A key tool is continuous amortization [13], but recently Cucker et al. [17] initiated a Smale-type average case analysis for subdivision algorithms.

Acknowledgements

The author is deeply grateful for the feedback and bug reports from Michael Burr, Matthew England, Rémi Imbach, Juan Xu and Bo Huang.

References

1. Abbott, J.: Quadratic interval refinement for real roots. ACM Commun. Comput. Algebra **48**(1/2), 3–12 (Jul 2014). https://doi.org/10.1145/2644288.2644291, http://doi.acm.org/10.1145/2644288.2644291
2. Agrawal, A., Requicha, A.: A paradigm for the robust design of algorithms for geometric modeling. Computer Graphics Forum **13**(3), C/33–44 (1994), 15th Annual Conference and Exhibition. EUROGRAPHICS'94
3. Becker, R.: The Bolzano Method to Isolate the Real Roots of a Bitstream Polynomial. Bachelor thesis, University of Saarland, Saarbruecken, Germany (May 2012)
4. Becker, R., Sagraloff, M., Sharma, V., Xu, J., Yap, C.: Complexity analysis of root clustering for a complex polynomial. In: 41st Int'l Symp. Symbolic and Alge. Comp. pp. 71–78 (2016), iSSAC 2016. July 20-22, Wilfrid Laurier University, Waterloo, Canada.
5. Becker, R., Sagraloff, M., Sharma, V., Yap, C.: A near-optimal subdivision algorithm for complex root isolation based on Pellet test and Newton iteration. J. Symbolic Computation **86**, 51–96 (May-June 2018)
6. Bennett, H., Papadopoulou, E., Yap, C.: Planar minimization diagrams via subdivision with applications to anisotropic Voronoi diagrams. Eurographics Symposium on Geometric Processing **35**(5) (2016), sGP 2016, Berlin, Germany. June 20-24, 2016.
7. Bennett, H., Yap, C.: Amortized analysis of smooth quadtrees in all dimensions. Comput. Geometry: Theory and Appl. **63**, 20–39 (2017), also, in Proceedings SWAT 2014.
8. de Berg, M., Cheong, O., van Kreveld, M., Overmars, M.: Computational Geometry: Algorithms and Applications. Springer-Verlag, Berlin, revised 3rd edition edn. (2008)
9. Boissonnat, J.D., Teillaud, M. (eds.): Effective Computational Geometry for Curves and Surfaces. Springer (2007)
10. Braußе, F., Collins, P., Kanig, J., Kim, S., Konečný, M., Lee, G., Müller, N., Neumann, E., Park, S., Preining, N., Ziegler, M.: Semantics, logic, and verification of "exact real computation" (2019)
11. Brent, R.P.: Algorithms for minimization without derivatives. Prentice Hall, Englewood Cliffs, NJ (1973)
12. Burr, M., Krahmer, F.: SqFreeEVAL: An (almost) optimal real-root isolation algorithm. J. Symbolic Computation **47**(2), 153–166 (2012)
13. Burr, M., Krahmer, F., Yap, C.: Continuous amortization: A non-probabilistic adaptive analysis technique. Electronic Colloquium on Computational Complexity (ECCC) **TR09**(136) (December 2009), http://eccc.hpi-web.de/report/2009/136/
14. Burr, M.A.: Continuous amortization and extensions: With applications to bisection-based root isolation. J. Symb. Comput. **77**, 78–126 (2016). https://doi.org/10.1016/j.jsc.2016.01.007, http://dx.doi.org/10.1016/j.jsc.2016.01.007
15. Burr, M.A., Gao, S., Tsigaridas, E.: The complexity of an adaptive subdivision method for approximating real curves. In: 42Int'l Symp. Symbolic and Alge. Comp. (ISSAC). pp. 61–68. ISSAC '17, ACM, New York, NY, USA (2017). https://doi.org/10.1145/3087604.3087654, http://doi.acm.org/10.1145/3087604.3087654
16. Burr, M.A., Gao, S., Tsigaridas, E.: The complexity of subdivision for diameter-distance tests. J. Symbolic Computation (2019), to appear.
17. Cucker, F., Ergür, A.A., Tonelli-Cueto, J.: Plantinga-vegter algorithm takes average polynomial time. arXiv:1901.09234 [cs.CG] (2019)
18. Du, Z., Eleftheriou, M., Moreira, J., Yap, C.: Hypergeometric functions in exact geometric computation. In: V.Brattka, M.Schoeder, K.Weihrauch (eds.) Proc. 5th Workshop on Computability and Complexity in Analysis. pp. 55–66 (2002), malaga, Spain, July 12-13, 2002. In Electronic Notes in Theoretical Computer Science, 66:1 (2002), http://www.elsevier.nl/locate/entcs/volume66.html.
19. Du, Z., Yap, C.: Uniform complexity of approximating hypergeometric functions with absolute error. In: Pae, S., Park, H. (eds.) Proc. 7th Asian Symp. on Computer Math. (ASCM 2005). pp. 246–249 (2006)
20. Emiris, I.Z., Pan, V.Y., Tsigaridas, E.P.: Algebraic algorithms. In: Gonzalez, T., Diaz-Herrera, J., Tucker, A. (eds.) Computing Handbook, 3rd Edition: Computer Science and Software Engineering, pp. 10: 1–30. Chapman and Hall/CRC (2014)
21. Higham, N.J.: Accuracy and stability of numerical algorithms. Society for Industrial and Applied Mathematics, Philadelphia, second edn. (2002)
22. Hsu, C.H., Chiang, Y.J., Yap, C.: Rods and rings: Soft subdivision planner for **R**ˆ3 x **S**ˆ2. In: Proc. 35th Int'l Symp. on Comp. Geom.(SoCG 2019) (June 18-21, 2019), to appear. CG Week 2019, Portland Oregon. Also in arXiv:1903.09416.
23. Hsu, C.H., Chiang, Y.J., Yap, C.: Rods and rings: Soft subdivision planner for **R**ˆ3 x **S**ˆ2. In: Proc. 35th Int'l Symp. on Comp. Geom.(SoCG 2019) [22], to appear. CG Week 2019, Portland Oregon. Also in arXiv:1903.09416.
24. Imbach, R., Pan, V., Yap, C.: Implementation of a near-optimal complex root clustering algorithm. In: Proc. Int'l Congress on Mathematical Software. LNCS, vol. 10931, pp. 235–244. Springer (2018), 6th ICMS, Notre Dame University. July 24-27, 2018.
25. Imbach, R., Pouget, M., Yap, C.: Effective subdivision algorithm for isolating zeros of real systems of equations, with complexity analysis (2019), to appear, 21st CASC, Moscow

26. Kearfott, R.B.: Rigorous global search: continuous problems, vol. 13. Springer Science & Business Media (2013)
27. Kerber, M., Sagraloff, M.: Efficient real root approximation. In: Schost, É., Emiris, I.Z. (eds.) ISSAC. pp. 209–216. ACM (2011)
28. Kincaid, D., Cheney, W.: Numerical Analysis: Mathematics of Scientific Computing. Brooks/Cole, 3rd edn. (2002)
29. Ko, K.I.: Complexity Theory of Real Functions. Progress in Theoretical Computer Science, Birkhäuser, Boston (1991)
30. Lien, J.M., Sharma, V., Vegter, G., Yap, C.: Isotopic arrangement of simple curves: An exact numerical approach based on subdivision. In: Hong, H., Yap, C. (eds.) Mathematical Software – ICMS 2014. vol. LNCS 8592, pp. 277–282. Springer (2014), seoul, Korea, Aug 5-9, 2014
31. Lin, L., Yap, C.: Adaptive isotopic approximation of nonsingular curves: the parameterizability and nonlocal isotopy approach. Discrete and Comp. Geom. **45**(4), 760–795 (2011)
32. Lin, L., Yap, C., Yu, J.: Non-local isotopic approximation of nonsingular surfaces. Computer-Aided Design **45**(2), 451–462 (Oct 2012), symp. on Solid and Physical Modeling (SPM). U. of Burgundy, Dijon, France, Oct 29-31, 2012.
33. Mehlhorn, K., Schirra, S.: Exact computation with `leda_real` – theory and geometric applications. In: Alefeld, G., Rohn, J., Rump, S., Yamamoto, T. (eds.) Symbolic Algebraic Methods and Verification Methods. pp. 163–172. Springer-Verlag, Vienna (2001)
34. Moore, R.E.: Interval Analysis. Prentice Hall, Englewood Cliffs, NJ (1966)
35. Ó'Dúnlaing, C., Yap, C.K.: A "retraction" method for planning the motion of a disc. J. Algorithms **6**, 104–111 (1985), also, Chapter 6 in *Planning, Geometry, and Complexity*, eds. Schwartz, Sharir and Hopcroft, Ablex Pub. Corp., Norwood, NJ. 1987.
36. Pedersen, P.: Counting real zeros. In: Proc. Conf. Algebraic Algorithms and Error Correcting codes. pp. 318–332. No. 539 in LNCS, Springer-Verlag (1991)
37. Pedersen, P.: Counting Real Zeros. Ph.D. thesis, New York University (1991), also, Courant Institute Computer Science Technical Report 545 (Robotics Report R243)
38. Pedersen, P., Roy, M.F., Szpirglas, A.: Counting real zeros in the multivariate case. In: Eyssette, F., Galligo, A. (eds.) Computational Algebraic Geometry, Progress in Mathematics, vol. 109. Birkhäuser, Boston, MA (1993)
39. Plantinga, S., Vegter, G.: Isotopic approximation of implicit curves and surfaces. In: Proc. Eurographics Symposium on Geometry Processing. pp. 245–254. ACM Press, New York (2004)
40. Preparata, F.P., Shamos, M.I.: Computational Geometry. Springer-Verlag (1985)
41. Riley, K., Hopson, M., Bence, S.: Mathematical Methods for Physics and Engineering. Cambridge University Press, third edn. (2006)
42. Sagraloff, M., Yap, C.K.: A simple but exact and efficient algorithm for complex root isolation. In: Emiris, I.Z. (ed.) 36th Int'l Symp. Symbolic and Alge. Comp. pp. 353–360 (2011), june 8-11, San Jose, California.
43. Sagraloff, M., Yap, C.K.: An efficient exact subdivision algorithm for isolating complex roots of a polynomial and its complexity analysis (July 2009), submitted. Full paper from http://cs.nyu.edu/exact/ or http://www.mpi-inf.mpg.de/~msagralo/
44. Schuurman, P., Woeginger, G.: Approximation schemes: A tutorial. In: Möhring, R., Potts, C., Schulz, A., Woeginger, G., Wolsey, L. (eds.) Lectures in Scheduling (2007), to appear.
45. Sharma, V., Yap, C.: Near optimal tree size bounds on a simple real root isolation algorithm. In: 37th Int'l Symp. Symbolic and Alge. Comp.(ISSAC'12). pp. 319 – 326 (2012), jul 22-25, 2012. Grenoble, France.
46. Sharma, V., Yap, C.K.: Robust geometric computation. In: Goodman, J.E., O'Rourke, J., Tóth, C. (eds.) Handbook of Discrete and Computational Geometry, chap. 45, pp. 1189–1224. Chapman & Hall/CRC, Boca Raton, FL, 3rd edn. (2017)
47. Snyder, J.: Generative Modeling for Computer Graphics and CAD. Symbolic Shape Design Using Interval Analysis. Academic Press Professional, Inc., San Diego, CA, USA (1992)
48. Trefethen, L.N., Bau, D.: Numerical linear algebra. Society for Industrial and Applied Mathematics, Philadelphia (1997)
49. Ueberhuber, C.W.: Numerical Computation 2: Methods, Software, and Analysis. Springer, Berlin (1997)
50. Wang, C., Chiang, Y.J., Yap, C.: On soft predicates in subdivision motion planning. Comput. Geometry: Theory and Appl. (Special Issue for SoCG'13) **48**(8), 589–605 (Sep 2015)
51. Weihrauch, K.: Computable Analysis. Springer, Berlin (2000)
52. Xu, J., Yap, C.: Effective subdivision algorithm for isolating zeros of real systems of equations, with complexity analysis. In: 44th Int'l Symp. Symbolic and Alge. Comp. (2019), jul 15-18. Beihang University, Beijing.
53. Yap, C., Sagraloff, M., Sharma, V.: Analytic root clustering: A complete algorithm using soft zero tests. In: The Nature of Computation. Logic, Algorithms, Applications. LNCS, vol. 7921, pp. 434–444. Springer (2013)
54. Yap, C., Sagraloff, M., Sharma, V.: Analytic root clustering: A complete algorithm using soft zero tests. In: The Nature of Computation. Logic, Algorithms, Applications [53], pp. 434–444
55. Yap, C.: Soft subdivision search and motion planning, II: Axiomatics. In: Frontiers in Algorithmics. Lecture Notes in Comp. Sci., vol. 9130, pp. 7–22. Springer (2015), plenary talk at 9th FAW. Guilin, China. Aug. 3-5, 2015.

56. Yap, C., Luo, Z., Hsu, C.H.: Resolution-exact planner for thick non-crossing 2-link robots. In: Proc. 12th Intl. Workshop on Algorithmic Foundations of Robotics (WAFR '16) (2016), dec. 13-16, 2016, San Francisco. The appendix in the full paper (and arXiv from `http://cs.nyu.edu/exact/` (and `arXiv:1704.05123 [cs.CG]`) contains proofs and additional experimental data.
57. Yap, C.K.: On guaranteed accuracy computation. In: Chen, F., Wang, D. (eds.) Geometric Computation, chap. 12, pp. 322–373. World Scientific Publishing Co., Singapore (2004)
58. Yap, C.K.: In praise of numerical computation. In: Albers, S., Alt, H., Näher, S. (eds.) Efficient Algorithms, LNCS, vol. 5760, pp. 308–407. Springer-Verlag (2009)
59. Yu, J., Yap, C., Du, Z., Pion, S., Bronnimann, H.: Core 2: A library for Exact Numeric Computation in Geometry and Algebra. In: 3rd Proc. Int'l Congress on Mathematical Software (ICMS). pp. 121–141. Springer (2010), kobe, Japan. Sep 13-17, 2010. LNCS No. 6327.