Certified Complex Root Isolation via Adaptive Root Separation Bounds

MICHAEL SAGRALOFF[1*], MICHAEL KERBER[2], MICHAEL HEMMER[3]

[1,2,3] Max-Planck-Institut für Informatik, Saarbrücken, Germany
[msagralo|mkerber|hemmer]@mpi-inf.mpg.de

Abstract

We address the problem of *root isolation* for polynomial systems: for an affine, zero-dimensional polynomial system of N equations in N variables, we describe an algorithm to encapsulate all complex solutions into disjoint regions, each containing precisely one solution (called *isolating regions*). Our approach also computes the multiplicity of each solution. The main novelty is a new approach to certify that a set of computed regions is indeed isolating. It is based on an adaptive root separation bound obtained from combining information about the approximate location of roots and resultant calculus. Here we use simple subdivision method to determine the number of roots within certain regions. The resultant calculus only takes place over prime fields to avoid the disadvantageous coefficient growth in symbolic methods, without sacrificing the exactness of the output. The presented approach is complete for uni- and bivariate systems, and in general applies in higher dimensions as well, possibly after a coordinate change.

1 Introduction

Finding the roots of a zero dimensional polynomial system is a fundamental problem of numerous applications spread over several important areas, such as algebraic geometry, computer graphics and computer aided geometric design. In particular, the design of robust and certified algorithms demands for efficient methods that determine isolating regions for all roots of polynomial systems. Such methods should also be capable to handle non-simple roots.

This work is driven by the question: How can fast but unreliable root solving techniques be combined with symbolic computations in an efficient way, such that the overall result can be certified? We see our main contribution in providing a novel certification scheme in this context. Its main idea is to follow two threads of computation in parallel. Both threads only deliver incomplete information, but their combination is sufficient to certify the result of the method.

The first thread is inspired by elimination methods such as *Multivariate resultants* and *Groebner bases*. Both are well-studied tools to obtain the solution set of a system with respect to a projection direction. However, both methods lead to polynomials with very large bitsizes (for intermediate results as well as for the final result), which causes a severe drawback regarding the performance. Therefore, our method computes the multivariate resultant (with some hidden variable) only in several prime fields $\mathbb{Z}_p$ and completely avoids Chinese Remaindering. In particular, all symbolic computations are performed using single precision arithmetic [4]. This method yields a lower bound on the number of projected solutions. Although this bound is very likely to match the exact number in practice, this cannot be certified without further knowledge.

The second thread follows an *Exclusion and Subdivision method*. It keeps on subdividing regions that may contain solutions (from now on, we call such connected regions *clusters*), whereas regions that doubtlessly do not contain a solution are discarded. As a simple exclusion method we use interval arithmetic. Usually, this is combined with a criterion to ensure that a cluster contains

*Correspondence to: Michael Sagraloff, Max-Planck-Institut für Informatik, Departement 1: Algorithms and Complexity, Campus E1 4, 66123 Saarbrücken, Germany; Tel: +49 681 9325 106, Fax: +49 681 9325 199

precisely one simple root, but these criteria mostly fail in the presence of multiple roots. Instead, we introduce two new methods based on homotopy arguments. The first ensures the presence of at least one root inside a cluster, the second sums up the multiplicities of the roots inside a cluster. Clearly, this cannot suffice to ensure that a cluster is isolating.

Our algorithm merges both building blocks to certify that clusters are isolating. For simplicity, we sketch the certification idea only for a univariate polynomial f (the general case is discussed in Section 2). Two real values are computed: LB, obtained by the modular symbolic computation, and UB, obtained by the distances between clusters, diameters of clusters, and their multiplicities. If the clusters are not isolating, LB (UB) defines a lower (upper) bound on the absolute value of the first non-vanishing subresultant coefficient* $\mathrm{sres}_k(f, f')$, which is essentially the product of squared root distances of f. During the certification, $LB \to \infty$ and $UB \to 0$. Once LB becomes larger than UB, it is proven by contradiction that the clusters are isolating.

We see a strength of our certification method in its adaptiveness to the concrete instance: The quality of the bounds LB and in particular UB are mainly determined by the size of $|\mathrm{sres}_k(f, f')|$, small values for it lead to faster certification. This adaptive behavior is a clear advantage compared to theoretical a priori bounds on $\mathrm{sres}_k(f, f')$ or on the separation of distinct roots which have to assume the worst-case.

Our new certification approach is embedded within a complete algorithmic description that takes a zero-dimensional system as input, and starts the subdivision on a sufficiently large bounding box. We decided for this setup for the sake of a comprehensive description, although there is no need to restrict to the proposed subdivision strategy. For instance, we propose that for an efficient realization of our ideas, a fast numerical solver (e.g, based on homotopy methods; see the section on related work) is first applied to approximate the solutions of the system, and our certification process comes in afterwards only to validate the outcome of the solver. We sketch the workflow of such a hybrid solver in the conclusion.

The presented method is complete for 2×2-systems and applies, in general, also to higher dimensional systems. It requires the multivariate resultant to be expressible by a Macaulay matrix. In unfortunate cases, this is not possible, even under projective transformations.

Related Work. Since polynomial root solving is such an important problem in several fields, plenty of distinct approaches exist and many textbooks are dedicated to this subject. See, for instance, [9, 22, 27, 29] for introductions to symbolic approaches such as (sparse) resultants, Groebner bases, and methods based on eigenvalue computations or on *rational univariate representation*.

Homotopy methods numerically track the continuous path of the known complex solutions of some trivial and appropriate polynomial system during a continuous deformation into the input system. Such methods, although very robust, lack the certification of their output in general. We recommend [26] for a more comprehensive overview.

Subdivision methods describe a further class of common tools. Algorithm of that kind profit from their efficiency and plainness. Most Implementations are using one of the numerous software packages for efficient interval arithmetic [2, 18, 20], such as IntBis, ALIAS, IntLab or MPFI. Alternative variants, using the Bernstein basis and convexity properties of their coefficients, have been addressed [19]. However, all these approaches lack to certify their results – in general, an approach stops when a certain subdivision depth is reached or each region contains a simple root, which can, for instance, be certified by the interval Newton test [26, Sec. 6.1]. So far, in case of multiple roots, all proposed methods have to go below a certain a-priori worst case root separation bound in order to certify that a region is isolating. As a result of the bad quality of these bounds based on their theoretical character, subdivision methods turn out to be impractical for an exact and complete

*In case of a univariate polynomial f. For the general case, we consider $|\mathrm{sres}_{k_j}(r_j, r_j')|$, where r_j is the elimination polynomial of $F = 0$ with respect to z_j and $k_j := \deg \gcd(r_j, r_j')$.

approach.

Also specializations of the general problem have been extensively studied. The probably most prominent one is univariate polynomial solving. In particular, this is an integral building block in elimination methods, where the (univariate) elimination polynomial has to be considered, and lifting the solutions to higher dimensions usually leads to further univariate systems.

Certified algorithms for real root isolation are mainly subdivision solvers based on Descartes' Rule of Signs or Sturm sequences, see [3, 10, 29] for modern approaches. For the complex case we refer to [16, 21, 23, 24, 25].

Other special cases of polynomial systems appear in real algebraic geometry. Recent implementations for computing the topology of algebraic curves either make use of elimination methods [11, 13] or subdivision [1, 7]. Already this low dimensional application shows the mentioned drawbacks of the distinct approaches: subdivision fails to give a certified result in degenerate cases whereas elimination methods suffer from costly symbolic computations.

Outline. We sketch our algorithm in Section 2. Therein, we refer to the Sections 3-6 for the details of its submethods. Section 7 finally concludes our results.

2 Our approach

We fix the following notation throughout the paper: our input system is given by polynomials $f_i \in \mathbb{Z}[z_1,\ldots,z_N]$ of corresponding total degrees d_i, $i = 1,\ldots,N$, and $D = \prod_{i=1}^{N} d_i$ defines their product. Then these polynomials induce a function $F : \mathbb{C}^N \to \mathbb{C}^N$ that maps a point $p \in \mathbb{C}^N$ to $(f_1(p),\ldots,f_N(p))$. By assumption, F has only finitely many isolated solutions (or roots), let $\Gamma = V(F)$ denote them. For $j = 1,\ldots,N$, let $\pi_j : \mathbb{C}^N \to \mathbb{C}, (z_1,\ldots,z_N) \mapsto z_j$ denote the canonical projection map with respect to z_j. Likewise, $\Gamma_j := \pi_j(\Gamma)$ is the set of z_j-coordinates of solutions of F.

Given F, our algorithm computes disjoint *isolating clusters* $C_1,\ldots,C_s \subset \mathbb{C}^N$, that is, each cluster C_i contains precisely one root of F and $\Gamma \subset \bigcup_{i=1}^{s} C_i$. In general, we will use the term *cluster* for a connected subset of $\mathbb{C}^N$.

`Transformation phase` (Section 3.1): As a first step, the algorithm ensure several genericity conditions[†] which further steps rely on. Matrices $M_1,\ldots,M_N$ with $M_j \in \mathbb{Z}[z_j]^{\ell_j \times \ell_j}$ (for some ℓ_j) are computed whose determinants $r_j \in \mathbb{Z}[z_j]$ satisfy the following properties: r_j describes the projected solutions of F with respect to the coordinate z_j, in short $V(r_j) = \Gamma_j$, and multiplicities are preserved under projection. Moreover, each r_j must be of degree D; this ensures that F has no solution at infinity, that is, all solutions of its homogenization are contained in $\mathbb{C}^N$, considered as an affine chart of $\mathbb{P}^N$. All these properties are tried to be ensured only using modular arithmetic (i.e., without computing the r_j's exactly). If this fails, the algorithm starts over with a transformation of F by a linear projective change of coordinates. Section 3.1 describes the details of the transformation phase.

The actual isolation algorithm depends on the following three subroutines.

`Guess_`k_j (Section 3.2): For $j \in \{1,\ldots,N\}$, it computes an upper bound k'_j for $k_j := \deg \gcd(r_j, r'_j)$, which also yields a lower bound m'_j on the cardinality $m_j = D - k_j$ of Γ_j. This is done by computing the index of the first non-vanishing principal subresultant coefficient of r_j and r'_j in a modular domain $\mathbb{Z}_p$, where the prime p is newly chosen in each call.

`Subdivide` (Section 4): Returns a set of disjoint clusters that cover all roots of F, and each contains at least one root (we call clusters known to contain at least one root *zero-clusters*). This is done by subdivision in $\mathbb{C}^N \cong \mathbb{R}^{2N}$ to exclude regions without roots, combined with a criterion to

[†] for $N = 1$, the phase can be skipped

identify zero-clusters. Repeated calls of `Subdivide` shrink the zero-clusters, and if a zero-cluster is not isolating, it will split into several parts after sufficiently many steps.

`Mult_of_clusters` (Section 5): For the zero-clusters C returned by `Subdivide`, the number $\mu(C)$ of roots of F inside each cluster is computed, counted with multiplicity (notice that $\mu(C) \geq 2$ does not imply that C is non-isolating, since multiple roots can occur). The method slightly perturbs F into $\tilde{F}$ such that roots remain in their zero-clusters, and such that a root of multiplicity μ turns into μ simple roots nearby. The number of roots inside a zero-cluster is than counted by further subdividing the cluster.

`Main routine:` Our main root isolation routine passes through two phases. First, the *synchronization phase* (Section 6.1) calls `Guess_k`$_j$ repeatedly for each projection direction to obtain a lower bound m'_j on m_j, Though it is very likely to coincide with m_j, m'_j might improve in further calls. In parallel, it calls `Subdivide` repeatedly to get smaller and smaller zero-clusters $C_1, \ldots, C_{s'}$. For each direction it examines $\pi_j(\bigcup C_i)$, which decomposes into connected components, called *projected clusters*. Once the number of projected clusters under π_j coincides with m'_j for each j, `Mult_of_clusters` is called for $C_1, \ldots, C_s$ (this also yields multiplicities of projected clusters), and the algorithm switches to the *certification phase*, described next.

The clusters $C_1, \ldots, C_{s'}$ are isolating under the condition that $m'_j = m_j$, or equivalently $k'_j = k_j$, for each j. The *certification phase* (Section 6.2) tries to verify this. It computes the values LB_j, that is, the product of the primes used in `Guess_k`$_j$ so far, and UB_j which is determined by the diameters and multiplicities of projected clusters, and the distances between them. If $k'_j > k_j$, LB_j (UB_j) is a lower (upper) bound for $|\mathrm{sres}_{k_j}(r_j, r'_j)|$. The algorithm again calls `Guess_k`$_j$ repeatedly, which makes LB_j arbitrary large, and in parallel, it calls `Subdivide` which lets UB_j converge to zero. Either, $LB_j > UB_j$ at some point which algorithmically proves by contradiction that $k'_j = k_j$. Or, a call of `Guess_k`$_j$ improves the upper bound on k_j, or a call of `Subdivide` makes a projected cluster split into two parts. Both cases disprove $k'_j = k_j$ and thus, clusters were not isolating yet; the algorithm then switches back to the synchronization phase.

3 Symbolic tools

3.1 *Transformation phase*: Multivariate Resultant [‡]

Crucial for our method is the knowledge of univariate polynomials $r_1, \ldots, r_N$ such that the roots of r_j are precisely the z_j-coordinates of points in Γ. Each r_j is represented as the determinant of a matrix M_j in the coefficients of $f_1, \ldots, f_N$. Also, each r_j should have degree D, which ensures that all D solutions of the homogenized system over $\mathbb{P}^N$ lie in the considered affine chart.

It is possible that such r_j's (more precisely, the M_j's) only exist after a projective coordinate transformation, and in degenerate instances our method might even fail completely to compute such M_j's. We closely follow the ideas described in [9, § 3.5], using multivariate resultants and the "hidden variable" approach. Here, we just mention the main facts from the theory; see [9, 12] for further explanations.

We first introduce the (affine) multivariate resultant:

Definition 1. *For a system of n polynomials* $(f_1, \ldots, f_n)$ *in* $n-1$ *variables,* $\mathrm{res}_{(f_1,\ldots,f_n)}$ *is an irreducible polynomial in the coefficients of the* f_i*'s. The resultant vanishes if and only if the homogenized system has a solution over* $\mathbb{P}^n$.

To apply the multivariate resultant in our problem with N equations in N variables, we consider one variable z_j as a parameter, that means, the system has coefficients in $\mathbb{Z}[z_j]$. Then, $\mathrm{res}_j :=$

[‡] for $N = 1$, the whole section can be skipped

$\mathrm{res}^{z_j}_{(f_1,\ldots,f_N)}$, $j=1,\ldots,N$, defines a polynomial in z_j. From the definition, it follows that its roots are precisely the z_j-coordinates where $F=0$ has a solution.

Theorem 2. *Let* $\deg(\mathrm{res}_j)=D$ *for all* $j=1,\ldots,N$. *Then* F *has precisely* D *roots in* $\mathbb{C}^N$, *counted with multiplicity*[§]. *For each* $(z_1,\ldots,z_N)\in\Gamma$, z_j *is a root of* res_j *and the multiplicity of* z_j *is the sum of the multiplicities of the points in its fiber.*

Proof. The degree of each res_j is bounded by D (cf. [12]). If any solutions is not finite, at least one res_j must have a "root at infinity", thus its degree drops by at least one, which shows (1). The second claim follows directly from the definition of the multivariate resultant and the last from the previous, noting that the resultant is continuous in the coefficients of the system, and that a slight perturbation of the system yields D simple solutions with distinct z_j-coordinates. □

In our algorithm, we exploit that res_j can be represented, at least generically, and up to a constant, as the determinant of a coefficient matrix of $(f_1,\ldots,f_N)$. This follows from a theorem due to Macaulay [17].

Theorem 3. *There exist matrices* M_j *and* M'_j *whose entries are polynomials in the coefficients of the system* $(f_1,\ldots,f_N)$ *and in* z_j, *such that* $\mathrm{res}_j=\frac{\det M_j}{\det M'_j}$. M'_j *does not contain* z_j.

For the definition of M_j and M'_j, see [9]. Theorem 3 implies that if for our concrete system $\det M'_j\in\mathbb{C}$ does not vanish, then $r_j:=\det M_j$ equals res_j, up to a constant.

At several places in our algorithm we need to upper bound the coefficients of r_j. Using the Hadamard bound on M_j, one obtains the following estimation:

Lemma 4. *The bitsize of the coefficients of each* r_j *is at most* $\sigma:=n(\tau+\log n)+\log(dn+1)$, *where* τ *is the maximal bitsize of any coefficient of* $(f_1,\ldots,f_N)$, $n=\binom{\sum_{i=0}^N d_i}{N-1}$ *and* $d=\max d_i$.

Proof. The dimension of M_j equals $\binom{\sum_{i\neq j} d_i}{N-1}<n$ (cf. [9]). Moreover, each entry of M_j is a univariate polynomial whose coefficients have bitsize at most τ, and whose degree is bounded by d. Using [3, Prop. 8.12], the determinant polynomial r_j has thus a bitsize bounded by σ. □

We next describe the transformation phase of the algorithm. All computations are performed modulo a prime number p. Let $\bar{\cdot}$ denote the operation that maps integers to their modular image in $\mathbb{Z}_p$. This map extends to integer polynomials and integer matrices in the obvious way.

Choose a random prime p and compute over $\mathbb{Z}_p$, for each $j=1,\ldots,N$, $\det\overline{M'_j}\in\mathbb{Z}_p$ and $\det\overline{M_j}\in\mathbb{Z}_p[z_j]$. If any determinant $\det M'_j$ vanishes, or for any j, $\deg(\det M_j)\neq D$, transform $(f_1,\ldots,f_N)$ via a random linear projective change of coordinates and start over with the transformed system.

We remark that the transformation phase might loop forever, in case that $\det M'_j$ vanishes for the input system, and for all its linear projective transformations.

For special cases of polynomial systems, res_j is explicitly known as determinant of a matrix, not just as a quotient. The most prominent is the case $N=2$, where the *Sylvester matrix* can be used; Theorem 3 and consideration of M'_j is not needed. Special cases with $N>2$ are discussed in [28].

A perhaps unpleasant feature of the transformation is that also infinite solutions of the original system are computed. It is possible to rule out these infinite solutions in a post-processing step

[§]The multiplicity of a root $\xi\in\mathbb{C}^N$ of F is defined as the dimension of the localization of $\mathbb{C}[z_1,\ldots,z_N]/(f_1,\ldots,f_N)$ at ξ considered as a $\mathbb{C}$-vector space (cf.[3, p.148])

after the algorithm. For that we consider a finite region that contains all finite roots of the original coordinate system. For instance, it is possible to consider the box $[-2^\sigma, 2^\sigma]^{2N}$ with σ from Lemma 4.

3.2 *Guess_k$_j$*: Modular computation

We turn to the method `Guess_k`$_j$ that computes an upper bound k'_j on $k_j = \deg\gcd(r_j, r'_j)$. Since $\deg(r_j) = D$ (guaranteed by the transformation phase), $D - k'_j$ constitutes a lower bound m'_j on m_j, the number of distinct roots of r_j.

To compute k'_j, we exploit the relation (e.g. [3, Prop. 4.25])

$$k_j := \deg\gcd(r_j, r'_j) = \min\{i \geq 0 \mid \mathrm{sres}_i(r_j, r'_j) \neq 0\},$$

where $\mathrm{sres}_i(f, g)$ denotes the i-th principal subresultant coefficient of f and g.

By definition, $\mathrm{sres}_i(r_j, r'_j)$ can again be expressed as determinant in the coefficients of r_j. Therefore, its computation is possible in a modular domain $\mathbb{Z}_p$ for a prime p. Defining λ_j as the leading coefficient of r_j, we obtain the following.

Proposition 5. *Let $j \in \{1, \ldots, N\}$, and p a prime, that does not divide λ_j or $D \cdot \lambda_j$, the leading coefficient of r'_j. Then, for $i = 0, \ldots, D$ we get* $\mathrm{sres}_i(\overline{r_j}, \overline{r'_j}) = \overline{\mathrm{sres}_i(r_j, r'_j)}$.

It follows directly

Lemma 6. *Let $p_1, \ldots, p_s$ be distinct prime numbers that do not divide $D\lambda_j$, and*

$$k'_j := \min_{\ell=1,\ldots,s} \min\{i \geq 0 \mid \mathrm{sres}_i(\overline{r_j}, \overline{r'_j}) \neq 0\},$$

where $\bar{\cdot}$ is the modular operation with respect to p_ℓ. Then, $k'_j \geq k_j$, and thus r_j has at least $m'_j := D - k'_j$ distinct complex roots. Moreover, for each $i \in \{0, \ldots, k'_j - 1\}$,

$$\mathrm{sres}_i(r_j, r'_j) = 0 \vee |\mathrm{sres}_i(r_j, r'_j)| \geq \prod_{\ell=1}^{s} p_\ell.$$

The second part of Lemma 6 constitutes a lower bound for the size of non-vanishing principal subresultants coefficients. We will exploit this lower bound in the certification phase of the main algorithm (Section 6.2).

We explain our method *Guess_k$_j$* next. For any $j \in \{1, \ldots, N\}$ it outputs a pair an upper bound $k_j^{(p)}$ for k_j, and the prime p that has been used to obtain this bound.

Choose a prime p not considered so far, compute $\overline{r_j} = \det\overline{M_j}$ and $\overline{r'_j}$, with respect to p, until $\deg\overline{r_j} = D$ and $\deg\overline{r'_j} = D - 1$. Compute $k_j^{(p)} = \min\{i \geq 0 \mid \mathrm{sres}_i(\overline{r_j}, \overline{r'_j}) \neq 0\} = \deg\gcd(\overline{r_j}, \overline{r'_j})$, and return the pair $(k_j^{(p)}, p)$.

Note that `Guess_k`$_j$ performs all computations in the domain $\mathbb{Z}_p$, no exact evaluation of r_j is necessary. The price we pay is that we have to cope with the uncertainty whether $k'_j = k_j$ holds or not. This guess must be checked at the end of the overall algorithm. However, we claim that a wrong guess is very unlikely since $k_j = k'_j$ will hold as soon as a prime p is chosen that does not divide $\mathrm{sres}_{k_j}(r_j, r'_j)$.

4 *Subdivide*: Subdivision

We apply a subdivision scheme on $\mathbb{C}^N \cong \mathbb{R}^{2N}$ to identify clusters containing roots of F. Writing each $z_j := x_j + i \cdot y_j$ and $f_j = g_j + i \cdot h_j$, F can be interpreted as a function $F : \mathbb{R}^{2N} \to \mathbb{R}^{2N}$ that maps a point $p = (x_1, y_1, \ldots, x_N, y_N) \in \mathbb{R}^{2N}$ to $(g_1(p), h_1(p), \ldots, g_N(p), h_N(p))$. For a box $A = [a_1, b_1] \times [c_1, d_1] \ldots \times [a_N, b_N] \times [c_N, d_N]$, let $\square F(A)$ be the result of evaluating F at A in interval arithmetic, e.g., by the use of the recursive Horner scheme or centered box evaluation (also denoted as modified affine arithmetic [14]). By the properties of interval arithmetic [20], we get $\mathrm{Im}(\mathrm{F}|_{\mathrm{A}}) \subset \square F(A)$. We call a box A *hot*, if $\square F(A)$ contains the origin. Clearly, non-hot boxes do not contain any root of F.

We start with an initial box containing all roots.[¶] In each iteration, every hot box B is subdivided into 2^{2N} even parts, which replace the old box B. All new boxes are tested to be hot, non-hot boxes are removed. In each state the hot boxes can be grouped into maximal connected regions, called *hot clusters*. In each iteration, a hot cluster either splits into smaller hot clusters, dies (i.e., vanishes completely), or persists, that means, it remains connected after the subdivision step. However, it is not clear whether a hot clusters indeed contains a root. We derive a method to ensure the presence of at least one root inside a hot cluster next.

Theorem 7. *Let $D \subset \mathbb{C}^N$ be an open, connected subset. If there exists a point $p \in D$ such that $|F(p)| < m_{\partial D} := \min_{\gamma \in \partial D} |F(\gamma)|$, then F has a root in D.*

Proof. Consider the parameterized function $F_t = F - tF(p)$ for $t \in [0,1]$. Then the roots of F_t continuously depend on t and F_1 has a root, namely p, within D. When passing from F_1 to $F = F_0$ this root continuously transforms into a root p' of F. If p' is outside D, then there must exist a $t_0 \in [0,1]$ such that F_{t_0} has a root p^* on ∂D. But, $0 = |F_{t_0}(p^*)| = |F(p^*) - t_0 F(p)| \geq ||F(p^*)| - |t_0 F(p)|| \geq m_{\partial D} - |F(p)| > 0$, thus p' is also located in D. □

Definition 8. *Let C be a hot cluster consisting of hot boxes $B_1, \ldots, B_s$. We define $\Delta(C)$ as the union of all boxes $\tilde{B}_1, \ldots, \tilde{B}_r$ that are adjacent to C. We define*

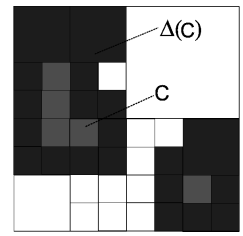
$$\varepsilon(C) := \min_{j=1\ldots,s} \left| F(\rho_j)) \right|$$

where ρ_j denotes the midpoint of B_j. Furthermore

$$\delta(C) := \min_{l=1,\ldots,r} \left| \square F(\tilde{B}_l) \right| > 0,$$

if $\partial C \subset \partial \Delta(C)$ and $\delta(C) := 0$, otherwise.

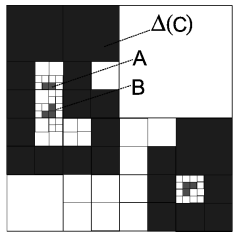
Notice that $\partial C \subset \partial \Delta(C)$ exactly holds iff none of the boxes B_j has a common point with the boundary of the initial box. In this situation we get $\min_{\gamma \in \partial C} |F(\gamma)| \geq \delta(C) > 0$. Hence, by Theorem 7, we can conclude that a cluster C contains a root of F if $\varepsilon(C) < \delta(C)$. For a sequence of clusters C_k that approximates a root ξ it is clear that $\varepsilon(C_k) \to 0$ as well as $\delta(C_k) \to 0$ while $k \to \infty$. This does not imply that we reach a state such that $\varepsilon(C_k) < \delta(C_k)$. But for some fixed k' there is a $k'' \geq k'$ such that $\varepsilon(C_{k''}) < \delta(C_{k'})$



Corollary 9. *Let C' and C be two hot clusters and C' be a descendant cluster of C, that is, $C' \subset C$. If $\varepsilon(C') < \delta(C)$ then C contains a root of f.*

[¶] e.g., one can use the box $[-2^\sigma, 2^\sigma]^{2N}$ with σ as in Lemma 4

To illustrate how Corollary 9 is applied, consider the picture on the right. Assume that the cluster C splits during the subdivision, and yields two hot sub-clusters A, B. If $\varepsilon(A) < \delta(C)$, then C must contain a root. However, it does not imply that A also contains a root, it only follows that $A \cup B$ contains a root.



We introduce a data structure that will help to identify clusters that are certified to contain a root of F. For that, we maintain a tree $\mathscr{T}$, called *cluster tree*, where each node v_C in $\mathscr{T}$ corresponds to a hot cluster C and a node $v_{C'}$ is a child of v_C if $C' \subset C$ is a hot cluster of the next iteration step. The root of $\mathscr{T}$ corresponds to the initial box. Moreover, each node in the tree maintains a δ and an ε-value that are initialized according to Definition 8. While δ does not change, new descendent nodes can give rise to better ε-values which are propagated to all their ancestors. Once $\varepsilon < \delta$ for a node, the corresponding cluster is known to contain a root by Theorem 7. We tag such a node and all its ancestors with a *zero* flag. We call these nodes *zero-nodes*, and the corresponding clusters *zero-clusters*. It may also happen that all descendent nodes of a node v die out. In this case the subtree which is rooted at v is completely removed from $\mathscr{T}$. Hence, the leaves of $\mathscr{T}$ are in one-to-one correspondents to the hot clusters in the current subdivision state.

Proposition 10. *A hot cluster that contains a root of F will eventually be a zero-cluster in the cluster tree. A hot cluster that does not contain a root of F will eventually be removed from the cluster tree.*

Moreover, we maintain a subtree $\mathscr{T}'$. We start with the root node of $\mathscr{T}$ and grow $\mathscr{T}'$ as follows. Add all children of a leaf of $\mathscr{T}'$ as soon as all of them are zero-nodes. By construction, the leaves of $\mathscr{T}'$ correspond to a set of disjoint zero-clusters, which covers all roots of F. We call this set the current *minimal zero-cluster overlay* (*MZCO*).

Proposition 11. *The* MZCO *constitutes a set of isolating clusters for F after sufficiently many subdivisions.*

A call of the method `Subdivide` triggers another subdivision step, updates $\mathscr{T}$, $\mathscr{T}'$ and finally returns the current *minimal zero-cluster overlay*.

---

5 *Mult_of_clusters*: Perturbation

The multiplicity of a root $\xi \in \mathbb{C}^N$ of F is defined as the dimension of the localization of the ring $\mathbb{C}[z_1,\ldots,z_N]/(f_1,\ldots,f_N)$ at ξ, considered as a $\mathbb{C}-$vector space (cf.[3, p.148]). A more intuitive description states that ξ is a root of multiplicity μ if and only if there exists a neighborhood $U(\xi) \subset \mathbb{C}^N$ such that almost any sufficiently small perturbation of F has exactly μ distinct, simple roots in $U(\xi)$.

We next discuss `Mult_of_clusters` that computes the sum of the multiplicities of all roots inside each $C \in \mathscr{C}$, where $\mathscr{C}$ is a MZCO as returned by the method `Subdivide`.

We first choose a *perturbation vector* $\upsilon \in \mathbb{Q}^{2N}$ such that $\forall C \in \mathscr{C} : |\upsilon| < \delta(C)$. The following Lemma ensures that roots can not leave their cluster when perturbing by υ.

Lemma 12. *Let C be some hot cluster and $\upsilon \in \mathbb{R}^{2N}$ with $0 < |\upsilon| < \delta(C)$, then the number of roots of $F + \upsilon$ equals the number of roots of F in C.*

Proof. Consider the parameterized function $F_t = F + t\upsilon$ and apply the same argumentation as in Theorem 7. □

We also ensure that υ is chosen such that all roots of $\tilde{F}$ are simple. One possibility is to check whether $\mathrm{res}(\tilde{r}_1, \tilde{r}_1') \neq 0$ over some $\mathbb{Z}_p$, p a random prime.[‖] In case of a failure we simply choose another υ and perform the modular computation with some other prime until we succeed.

Once υ is chosen, we know that all roots of $\tilde{F}$ are simple. In particular, we know that $\tilde{F}$ has precisely D simple roots. This is guaranteed by the fact that F, and therefore $\tilde{F}$, has only finite roots, see also Section 3.1. Hence, we can apply a variant of our subdivision scheme to $\tilde{F}$ as follows:

For a given MZCO $\mathscr{C} := \{C_1, \ldots, C_s\}$ for F the method `Mult_of_clusters` computes $\tilde{F}$ for some proper perturbation vector $\upsilon \in \mathbb{Q}^{2N}$. Now the subdivision method as discussed in Section 4 is applied to $\tilde{F}$ with initial clusters $C_1, \ldots, C_s$ inducing a forest of cluster trees. The subdivision is continued until the MZCO defined by the forest precisely contains D clusters. The number of zero-clusters in the *MZCO* with root C_i then gives the number $\mu(C_i)$ of roots of F in the cluster C_i, counted with multiplicities.

Note that the method `Mult_of_clusters` can be modified such that it uses a separate perturbation vector υ_i for each cluster $C_i \in \mathscr{C}$. That is, each υ_i fulfills $|\upsilon_i| < \delta(C_i)$ and all roots of $F + \nu_i$ within C_i are simple.

Improved Perturbation Vectors: So far, we sketched a method to compute a global, randomly chosen perturbation vector υ. Although it is already guaranteed that $\tilde{F}$ has only simple roots, an unfortunate choice of υ leads to a bad root separation, and hence to a high subdivision depth for $\tilde{F}$. To overcome this problem, we next propose an alternative approach. For each $C_i \in \mathscr{C}$, it computes a perturbation vector υ_i independently. It makes use of a criterion to guarantee that an axis-aligned box contains at most one root, which is simple. We introduce some notation first.

It is well known (see [3, Prop. 4.94] for a proof) that the Jacobian $J_F := (\partial f_j / \partial z_k)_{1 \leq j,k \leq N}$, of F at ξ has full rank if ξ is a simple root of F. For its real counter part

$$D_F := \begin{pmatrix} \frac{\partial g_j}{\partial x_k} & \frac{\partial g_j}{\partial y_k} \\ \frac{\partial h_j}{\partial x_k} & \frac{\partial h_j}{\partial y_k} \end{pmatrix}_{1 \leq j,k \leq N}$$

it holds that $\det(D_F) = \det(J_F)^2$ [15, p.27], thus it follows:

Lemma 13. *F has a simple root at $\xi \in \mathbb{C}^N$ if and only if $F(\xi) = 0$ and $\det(D_F)(\xi) = (\det(J_F)(\xi))^2 \neq 0$.*

For $n > 1$, Rolle's theorem is not directly applicable to functions $\varphi : \mathbb{R}^n \mapsto \mathbb{R}^n$, thus F might have two roots without a root of $\det(D_F)$ in between. However, we can use another criterion that exploits the behavior of interval arithmetic. We consider the matrix D_F and denote its row vectors by

$$G_j := \left(\frac{\partial g_j}{\partial x_i}, \frac{\partial g_j}{\partial y_i} \right)_{1 \leq j \leq N}, \quad H_j := \left(\frac{\partial h_j}{\partial x_i}, \frac{\partial h_j}{\partial y_i} \right)_{1 \leq j \leq N}$$

Then for any $\Phi := (p_1, \ldots, p_N, q_1, \ldots, q_N) \in \left(\mathbb{R}^{2N}\right)^{2N}$, we define $M(\Phi)$ as the $2N \times 2N-$matrix whose rows are the vectors $G_j(p_j)$ and $H_j(q_j)$, $j = 1, \ldots, N$.

Lemma 14. *Let $B \subset \mathbb{R}^{2N}$ be an axis-aligned box and $D : \left(\mathbb{R}^{2N}\right)^{2N} \to \mathbb{R}^{2N}$ defined by $D(\Phi) := \det(M(\Phi))$. Then*

[‖] For the definition and the computation of the resultant $\tilde{r}_1$ with respect to $\tilde{F}$ we refer to Section 3.

1. *If $0 \notin \square D(B^{2N})$, then B contains at most one root of F, which is simple.*
2. *If B contains exactly one root of F (counted with multiplicity) and B is sufficiently small, then $0 \notin \square D(B^{2N})$.*

Proof. We consider the case $N = 1$ where $F : \mathbb{C} \to \mathbb{C}$ is a univariate polynomial. Then the general case follows in a complete analog manner. F can be written as $F(x+iy) = f(x,y) + ig(x,y)$ with polynomials $f, g \in \mathbb{Z}[x,y]$. If $B \subset \mathbb{R}^2$ contains a multiple root a of F, then $D(a,a) = D_F(a) = 0$, thus we can restrict to the case where B contains two distinct roots a, b. We denote $[a,b]$ the line segment connecting a and b. From Rolle's theorem in several real variables, applied to the polynomials f and g, it follows the existence of points $p, q \in [a,b]$ with

$$\begin{pmatrix} f_x(p) \\ f_y(p) \end{pmatrix} \cdot (a-b) = \begin{pmatrix} g_x(q) \\ g_y(q) \end{pmatrix} \cdot (a-b) = 0.$$

Thus, $(f_x(p), f_y(p))^t$ and $(g_x(p), g_y(p))^t$ are perpendicular to $a-b$, which is only possible if they are linearly dependent. As a consequence we must get

$$\det \begin{pmatrix} f_x(p) & f_y(p) \\ g_x(q) & g_y(q) \end{pmatrix} = 0$$

Hence, it follows that $0 \in \square D(B^2)$. In contrast, if B contains only one simple root $a \in B$ then $\det(D_F|_p) \neq 0$. D is a continuous function, thus if we choose B small enough, this guarantees that $0 \notin \square D(B^2)$. □

Note the subtle difference of Lemma 14 compared with the criterion $0 \notin \square \det D_F(B)$ which does not guarantee the presence of at most one root in B; it is needed that the values in each row of the matrix are chosen independently of each other.

Using Lemma 14 we compute for each cluster $C \in \mathscr{C}$ a corresponding perturbation vector v_C. Compared to the modular approach, this leads to a better separation for the roots of $\tilde{F} := F + v_C$ within C. For a box B we define its expanded box B^+ as the box with the same center as B, and thrice as large side length. We call a box *simple*, if $0 \notin \square D((B^+)^{2N})$, and *non-simple* otherwise.

Copy all boxes of C that are non-simple into a new subdivision queue. Start subdividing, and keep only non-simple boxes in the queue. In addition, evaluate $\square F(B)$ for each box in the queue, let $U \subset \mathbb{C}^N$ be the union of these interval vectors. Stop the subdivision as soon as $\mathbb{R}^{2N} \setminus U$ contains a vector v with $|v| \in [0, \delta(C)]$. Set $v_C := v$ and return.

Why does the above algorithm terminate? Notice that $\det D_F = 0$ describes a hypersurface $V(det(D_F))$ in $\mathbb{C}^N$, thus the function values of F on $V(det(D_F))$ also describe a hypersurface in $\mathbb{C}^N$. It follows that after finitely many subdivision steps, the complement of the union U of interval vectors $\square F(B)$ must contain a vector v with $|v| \in [0, \delta(C)]$.

Lemma 15. *For a cluster C, let v_C be chosen as by the algorithm above, then $\tilde{F} := F + v_C$ has only simple roots within C and all these roots are separated by at least s, where s denotes the minimal side length of all boxes considered during the subdivision.*

Proof. Note first that $D\tilde{F} = DF$, thus (non-)simple boxes remain (non-)simple when switching to $\tilde{F}$. From the choice of v_C, it follows that each $\xi \in C$ with $F(\xi) = v_C$ is contained in one of the simple boxes. Hence, all roots of $\tilde{F}$ within C are simple as well. Now consider two roots a, b of $\tilde{F}$, then both of them are contained in simple boxes B_1 and B_2 respectively. As none of the boxes B_1 or B_2 contains two roots we get that $B_1 \neq B_2$ and B_1, B_2 are not adjacent. □

We remark another application of Lemma 14, although it is not directly needed for our application. Observe that the method *Mult_of_clusters* relies on the fact that all clusters are considered simultaneously, because it stops as soon as the total number of certified clusters equals D.

With a slight modification it is possible to describe a local version of *Mult_of_clusters* that computes the number of roots, counted with multiplicity, within a given cluster C. It computes a ΔC as before, and subdivides C with respect to the evaluation function $F_{\Delta C}$. Also, a cluster tree is maintained for C to identify zero-clusters, as before. For each zero-cluster $C_0 \subset C$ in the subdivision, a box B of minimal size is computed that completely contains this cluster. If $0 \notin \square D(B^{2N})$, the zero-cluster contains precisely one simple root, we call it *simple cluster*. If a cluster is simple, all its subclusters are simple as well. If no such box B can be computed, further subdivision is necessary. The algorithm stops if each element of the current MCZO is simple.

6 The main routine

6.1 *Synchronization phase*: Projection

In Section 3.2, we explained a method `Guess_k`$_j$ to obtain a lower bound m'_j on m_j, the number of distinct roots of r_j. In Section 4, we presented a method `Subdivide` to compute a set of disjoint clusters in $\mathbb{C}^N$, containing all roots of F, and each set contains at least one solution (we called such clusters *zero-clusters*). In the synchronization phase, both subroutines are combined to make their outcome coherent. For that, we introduce the notion of a *projected cluster*. Recall that, for all $j \in \{1,\ldots,N\}$, the roots of r_j coincide with Γ_j. Therefore, for a zero-cluster $C \subset \mathbb{C}^N$, $\pi_j(C) \subset \mathbb{C}$ is a connected region that contains at least one root of r_j. When projecting all clusters returned by `Subdivide`, some clusters might overlap in the projection. We therefore define

Definition 16. *Let $\mathscr{C} = \{C_1,\ldots,C_s\}$ be a set of zero-clusters returned by* `Subdivide`. *The* projected clusters *under π_j are the maximal connected components of $\bigcup_{C\in\mathscr{C}} \pi_j(C)$. The* multiplicity *$\mu(R)$ of a projected cluster R is defined as*

$$\mu(R) = \sum_{C\in\mathscr{C},\pi_j(C)\subset R} \mu(C),$$

where $\mu(C)$ is the multiplicity of the cluster C.

By the properties of r_j (Theorem 2), it follows

Lemma 17. *Let R be a projected cluster under π_j. Then R contains exactly $\mu(R)$ roots of r_j, counted with multiplicity. In particular, the set of projected clusters covers all roots of r_j and each projected cluster contains at least one root.*

During the subdivision, the clusters in $\mathbb{C}^N$ become arbitrary small, and the same holds for their projections. Thus, after sufficiently many calls of `Subdivide`, the returned clusters will induce exactly m_j projected clusters under π_j.

Lemma 18. *Let $\mathscr{C} = \{C_1,\ldots,C_s\}$ be a set of zero clusters returned by* `Subdivide`, *such that, for each j, there are m_j projected clusters under π_j. Then each cluster C_i contains exactly one root of F.*

Proof. If a cluster C_i contains two solutions, they differ in at least one variable z_j, and so, r_j has more than m_j distinct solutions, a contradiction. □

Our algorithm, however, does not apply Lemma 18 directly, since computing the m_j's is too costly. Instead, the synchronization method both computes a lower bound m'_j on m_j, and performs subdivision in $\mathbb{C}^N$, until there are precisely m'_j projected clusters with respect to z_j for each $j = 1,\ldots,N$. In the subsequent certification phase, it will be verified (or falsified) that $m'_j = m_j$ holds.

Here is the detailed description of the synchronization phase. The algorithm stores for each j a *current guess* m'_j, initially set to 0, and a number LB_j, initially set to one (the latter will be used in the certification). Also, it initially calls `Subdivide` and stores the returned set into $\mathscr{C}$.

Repeat the following steps: For each $j = 1,\ldots,N$, compute the projected clusters for $\mathscr{C}$ under π_j, let c_j denote their number. If $m'_j = c_j$ for all j, call `Mult_of_clusters` for each $C \in \mathscr{C}$, compute the multiplicity of each projected cluster according to Definition 16, and pass to the certification phase. Otherwise, let j be such that $m'_j \neq c_j$. If $m'_j < c_j$, call `Guess_k`$_j$, let $(k_j^{(p)}, p)$ be the result. Set m'_j to $\max\{m'_j, D - k_j^{(p)}\}$, set LB_j to $LB_j \cdot p$, and proceed with the next iteration. If $m'_j > c_j$, call `Subdivide`, set $\mathscr{C}$ to its output, and proceed with the next iteration.

Both m'_j and c_j are lower bounds for m_j, and after sufficiently many calls of `Guess_k`$_j$ or `Subdivide`, respectively, both will be set to m_j. However, the algorithm might jump to the certification phase earlier, when $c_j = m'_j \neq m_j$. As we will see, the certification phase will then falsify $m'_j = m_j$, and increase at least one of the values $c_1,\ldots,c_N,m'_1,\ldots,m'_N$.

6.2 *Certification phase:* Separation bounds

As Lemma 18 states, the clusters computed by the synchronization phase contain precisely one root of F under the condition that $m'_j = m_j$, or equivalently $k'_j = k_j$, for each j. For the sake of simpler notation in this subsection, we will fix one projection variable z_j, and set $k := k_j$, $k' := k_j$, $m := m_j$, $m' := m'_j$, and $r := r_j$. Note that $D := \deg(r)$. Also, we will denote the projected clusters for r by $R_1,\ldots,R_{m'}$. Our goal is to prove (or disprove) $k' = k$.

Our certification scheme uses two dynamically changing values $LB := LB_j$ and $UB := UB_j$ with the following properties:

- If $k' > k$, it holds that $LB \leq |\mathrm{sres}_k(r,r')| \leq UB$.
- If $k = k'$, further calls of `Guess_k`$_j$ improve LB. More precisely, a sequence of calls of `Guess_k`$_j$ leads to values LB that diverge to $+\infty$.
- If $k = k'$, further calls of `Subdivide` improve UB. More precisely, a sequence of calls of `Subdivide` leads to values UB that converge to 0.

The idea for the certification is to call `Guess_k`$_j$ and `Subdivide` simultaneously (or alternating), until $LB > UB$ which proves that $k' = k$. If a call of `Guess_k`$_j$ decreases k', or if any call of `Subdivide` leads to a split of some cluster R_i, the guess is falsified. In this case, the algorithm has to return to the synchronization phase to produce a new guess for k.

How do we obtain such bounds LB and UB? For LB, the answer is simple. Recall that Lemma 6 provides a lower bound for each non-vanishing $|\mathrm{sres}_l(r,r')|$ with $l < k' = k'_j$, namely the product of the primes considered so far by `Guess_k`$_j$. The algorithm keeps track of this product by the variable LB_j, compare the description of the synchronization phase. Clearly, each call of `Guess_k`$_j$ makes LB larger, and the product diverges to $+\infty$.

We turn to UB, for which we exploit our knowledge about the clusters of r, and the multiplicity of each. More precisely, for each such cluster R, we know the number $\mu(R)$ of roots of r inside R, counted with multiplicity (compare Lemma 17). To derive an adaptive upper bound for $|\mathrm{sres}_k(r,r')|$, we study the possible values of the root product of r.

Definition 19. *Given m' clusters $R_1,\ldots,R_{m'}$ for r, a set $\Psi := \{\psi_1,\ldots,\psi_n\} \subset \bigcup_{i=1,\ldots,m'} R_i$ with $n \geq m'$ is called a* valid root distribution of order *n, if the number of elements of V inside R_i is at least one, and at most $\mu_i := \mu(R_i)$. We also define the product*

$$P(\Psi) := \prod_{1\leq i<j\leq n} (\psi_i - \psi_j)^2.$$

The roots of r obviously define a valid root distribution of order m. We need some additional notation: For two clusters R_i and R_j of r, not necessarily distinct, define d_{ij} to be the maximal distance between a point in R_i to a point in R_j (for $i = j$, d_{ij} is the diameter of R_i). It follows directly

Proposition 20. *Let $\Psi := \{\psi_1,\ldots,\psi_{m'}\}$ be a valid root distribution of order m'. Then $P(\Psi) \leq \prod_{1\leq i<j\leq m'} d_{ij}^2$.*

Moreover, we can relate valid root distributions of order $n+1$ with valid root distributions of order n as follows.

Lemma 21. *Given m' clusters $R_1,\ldots,R_{m'}$ for r, and two valid root distributions $\Psi^{(n)} := \{\psi_1,\ldots,\psi_n\}$, $\Psi^{(n+1)} := \{\psi_1,\ldots,\psi_{n+1}\}$ of order n and $n+1$, respectively. Then $P(\Psi^{(n+1)}) \leq \beta^2 P(\Psi^{(n)})$ with*

$$\beta := \max_{i=1,\ldots,m'} \{\prod_{j=1}^{m'} d_{ij} \max\{d_{ij},1\}^{\mu_j - 1}\}$$

Proof. Notice that $P(\Psi^{(n+1)})$ can be written as the product of $P(\Psi^{(n)})$ and $\prod_i |\psi_i - \psi_{n+1}|^2$. Let c_i, $i = 1,\ldots,m'$, denote the number of elements among $\psi_1,\ldots,\psi_n$ inside the cluster R_i. By definition, $1 \leq c_i \leq \mu_i$ for all i. Further, let q be such that $\psi_{n+1} \in R_q$. Then, the additional factor $\prod_i |\psi_i - \psi_{n+1}|^2$ can be upper bounded by

$$\prod_{j=1}^{m'} \left(d_{qj}^{c_j}\right)^2 = \prod_{j=1}^{m'} \left(d_{qj} d_{qj}^{c_j - 1}\right)^2 \leq \left(\prod_{j=1}^{m'} d_{qj} \max\{1, d_{qj}\}^{\mu_j - 1}\right)^2.$$

□

Theorem 22. *Define*

$$UB := 2^{\sigma(2D-1)} \cdot e^{D/e} \cdot \beta^2 \cdot \max\{\beta,1\}^{2(k'-1)} \prod_{1\leq i<j\leq m'} d_{ij}^2,$$

with σ as in Lemma 4. If $k < k'$, then $\mathrm{sres}_k(r,r') \leq UB$.

Proof. Combining [10, Prop. 3.7] with [3, Prop. 4.27], we obtain the following equation for sres_k:

$$\mathrm{sres}_k(r,r') \quad = \quad \lambda^{2(D-k)-1} \prod_{i=1,\ldots,m} \mathrm{mult}(\phi_i) \cdot P(\Phi),$$

where $\Phi := \{\phi_1,\ldots,\phi_m\}$ are the roots of r, $\mathrm{mult}(\phi_i)$ is the multiplicity of ϕ_i, and λ is the leading coefficient of r.

We bound each factor separately. $\lambda^{2(D-k)-1} \leq 2^{\sigma(2D-1)}$ is obvious, as 2^σ is an upper bound for λ (cf. Lemma 4). For $\prod_i \mathrm{mult}(\phi_i)$, note that this is a product of $D-k$ numbers that sum up to D. One can show that such a product is bound by $\left(\frac{D}{D-k}\right)^{D-k}$, and by maximizing $\left(\frac{d}{x}\right)^x$, one easily verifies that its maximum is $e^{D/e}$.

For the last factor, reorder the roots Φ of r such that ϕ_i lies in cluster R_i, for $i = 1,\dots,m'$, the other $m-m'$ roots lie in arbitrary clusters. Since Φ is a valid root distribution of order m, $\Phi^{(n)} := \{\phi_1,\dots,\phi_n\}$ is a valid root distribution of order n for each $m' \le n \le m$. Applying Lemma 21 $(m-m')$ times, we obtain that $P(\Phi) \le \beta^{2(m-m')} P(\Phi^{(m')})$. Now the claim follows by Proposition 20, and by the fact that $\beta^{2(m-m')} = \beta^{2(k'-k)} = \beta^2 \beta^{2(k'-k-1)} \le \beta^2 \max\{1,\beta\}^{2(k'-k-1)} \le \beta^2 \max\{1,\beta\}^{2(k'-1)}$. □

Indeed, the bound as defined in Theorem 22 has the properties that we demanded for UB. As just shown, it is an upper bound for $|\mathrm{sres}_k(r,r')|$, if $k < k'$. Moreover, β, as defined in Lemma 21 becomes smaller when the clusters get smaller. If $k = k'$, the diameters d_{ii} all tend to zero, when `Subdivide` is repeatedly called. As each possible value of β contains at least one diameter d_{ii} as a factor, UB tends to zero, since all other quantities are non-increasing.

We next describe the certification phase. From the synchronization phase, the algorithm knows current guesses m'_j, and values LB_j, for each $j = 1,\dots,N$. Also, it has stored a set of clusters $\mathscr{C}$, which induce precisely m'_j projected clusters under π_j, and the multiplicity of each projected cluster is known.

For each $j = 1,\dots,N$, call simultaneously (or alternately) `Guess_k`$_j$ and `Subdivide`. When `Guess_k`$_j$ returns $(k_j^{(p)}, p)$, update the values m'_j and LB_j accordingly, as in the synchronization phase. If m'_j increases, the guess was wrong; switch back to the synchronization phase. After each call of `Subdivide`, update $\mathscr{C}$ and the projected clusters under π_j. If the number of projected clusters has been increased, the guess was wrong; switch back to the synchronization phase. Otherwise, compute UB_j as in Theorem 22. If $LB_j > UB_j$, it is certified that $m'_j = m_j$; proceed with the next j. When all j's have been considered, return the isolating clusters $\mathscr{C}$.

In case of a wrong guess m'_j, the certification phase not just falsifies $m'_j = m_j$, but also increases either m'_j itself, or the number of projected clusters under π_j. Consequently, the certification phase is never called twice for the same guesses $(m'_1,\dots,m'_N)$. Since the guess can only increase finitely often, this shows that the algorithm eventually terminates. However, we remark again that it is rather unlikely that the algorithm switches back to the synchronization phase at all, as we expect in practice that already the first call of `Guess_k`$_j$ will yield k_j as result.

Remark. The bound $UB = UB_j$ used by the algorithm certainly has room for optimizations. We hint at an alternative bound, based on the following result.

Lemma 23. *For a set $\mathscr{R}$ of projected clusters $R_1,\dots,R_{m'}$ and maximal distances d_{ij} between R_i and R_j, consider*

$$M(\mathscr{R}) := \max \lambda^{2n-1} \cdot \prod_i \prod_{1\le l\le n_i} \mu_{i,l} \cdot \prod_i d_{ii}^{n_i(n_i-1)} \prod_{1\le i<j\le n} d_{ij}^{n_i n_j}$$

where the maximum is taken over all n ($m' < n \le D$) and all possible values $n_i, \mu_{il} \ge 1$, $i = 1,\dots,m'$, and $l = 1,\dots,n_i$ such that $\sum_i n_i = n$ and $\sum_l \mu_{il} = \mu(R_i)$. If $k < k'$, $M(\mathscr{R})$ constitutes an upper bound of $|\mathrm{sres}_k(r,r')|$.

The estimation follows by considering a valid root distribution $\Psi^{(n)}$ of order n such that within each R_i there are exactly n_i elements of $\Psi^{(n)}$ which we assign multiplicities μ_{il}. We omit a more detailed proof for brevity.

For an improved UB, each factor of the product is upper-bounded. The factor $\prod_i \prod_l \mu_{i,l}$ can easily be bounded by $(D/n)^n$. The factor involving the d_{ij}'s leads to a quadratic convex optimization problem in the variables n_i whose matrix becomes diagonal dominant for sufficiently small clusters.

Obviously, such an UB is computationally more involved compared to Theorem 22, but the sharper upper bound might amortize this additional cost.

7 Conclusion and further work

We have described a novel certification scheme that allows to certify a collection of regions in $\mathbb{C}^N$ as isolating for the solutions of a zero-dimensional polynomial system over $\mathbb{C}^N$. Our approach combines the advantages of subdivision and modular symbolic computation. The output is certified by homotopy arguments and bounds on subresultants. We emphasize that an exact evaluation of resultants or Groebner bases is not necessary, that is, we only perform modular computations without lifting the elimination polynomials to $\mathbb{Z}$. Thus, all (intermediate) results are kept handy during the computation. At the same time, the performance of the proposed method adaptively depends on several magnitudes in the algorithm, such as the separation of roots and the size of $\mathrm{sres}_{k_j}(r_j, r'_j)$, instead of using worst-case bounds for them. To the best of our knowledge, only theoretical worst case bounds have been studied [5, 6, 8] so far. With respect to practical efficiency, these bounds are not applicable which results from the fact that, in all situation, the worst case scenario has to be taken into account. We consider our method as the first approach to introduce an adaptive root separation bound. On the one hand our method processes the exact information given by the integer coefficients of the polynomial system by the use of modular computation, and on the other hand, our bound directly depends on the actual geometric situation given by the roots of the system.

Our exposition in this paper is comprehensive in its essence, but does not mention all optimizations that an actual implementation should take care of. For instance, we expect that a more careful choice of the bound UB_j, as suggested at the end of Section 6.2, speeds up the certification phase. Recently, it was shown [23] that, for isolating the complex roots of a univariate polynomial of degree N and bitsize L, a subdivision approach based on centered box evaluation is quite effective. In the corresponding paper, the authors introduce a method called CEVAL that uses centered box evaluation as an exclusion predicate. They showed that the width of the subdivision tree does not exceed $O((N \log N)^2)$ boxes at each subdivision. Applying a different approach to certify the existence of exactly one root within a region, it was also proven that the overall method requires only $\tilde{O}(N^4 L^2)$ bit operations which matches the costs of most effective and exact methods for real root isolation. For higher dimensional systems no such results are available yet, but we are convinced that the techniques from the one dimensional case also apply to the more general setting. Our future research will concentrate on such an analysis.

For an efficient implementation we propose to use fast numerical methods to *find* approximations of the roots. As these methods do not provide any guarantees for their output we consider the role our subdivision methods as crucial in order to *certify* that a region contains (a certain number of) roots. We see a hybrid approach combining a fast numerical solver with our method: Therein, the numerical solver serves as a fast tool to achieve good approximations of the solutions. From our subroutines *Subdivide* and *Mult_of_clusters*, based on subdivision, we determine the exact number of roots within each of the obtained regions and check whether all solutions are captured. If this test fails, the numerical method is restarted with increased arithmetic precision arithmetic. With our *certification phase*, it is possible to verify that the regions are isolating. Thus, assuming that the numerical method determines arbitrary good approximations of all solutions for some sufficiently large precision, this shows the feasibility of a hybrid certified method to isolate all roots.

We further remark that all proposed methods are perfectly suited for parallel computations. For the modular computations this is due to the fact that unhandy terms are avoided and that it is possible to run a large number of distinct modular computations in parallel. The latter holds for the subdivision routines as well, since distinct clusters can be examined independently. We plan to

implement and benchmark our algorithm to answer the question whether these advantages lead to measurable effects also in practice.

Our algorithm requires the solution set of the input system to be zero-dimensional. Furthermore, its resultants must be computable over a prime field $\mathbb{Z}_p$. We achieve this by representing resultants as determinants of Macaulay matrices, but for $N \geq 3$ this might fail in unfortunate cases. This constitutes the only obstacle for our algorithm to be complete for higher dimensions. A natural question is whether our ideas also apply to non zero-dimensional systems, and whether a variation of the proposed method can guarantee to solve the system in all cases.

References and Notes

[1] L. Alberti, B. Mourrain, and J. Wintz. Topology and Arrangement Computation of Semi-Algebraic Planar Curves. *Computer Aided Geometric Design*, 25(8):631–651, 2008.

[2] G. Alefeld and J. Herzberger. *Introduction to Interval Computations*. Computer Science and Applied Mathematics. New York: Academic Press Inc., 1983.

[3] S. Basu, R. Pollack, and M.-F. Roy. *Algorithms in Real Algebraic Geometry*, volume 10 of *Algorithms and Computation in Mathematics*. Springer, 2nd edition, 2006.

[4] H. Brönnimann, I. Z. Emiris, V. Y. Pan, and S. Pion. Computing exact geometric predicates using modular arithmetic with single precision. In *SCG '97: Proc. of the 13th Ann. Symp. on Computational Geometry*, pages 174–182. ACM press, 1997.

[5] W. D. Brownawell and Chee K. Yap. Lower bounds for zero-dimensional projections. In *Proc. Int'l Symp. Symbolic and Algebraic Comp. (ISSAC'09)*, page To appear, 2009. KIAS, Seoul, Korea, Jul 28-31, 2007. DOI: http://doi.acm.org/10.1145/1277548.1277562. In press, Journal of Symbolic Computation.

[6] M. Burr, S.W. Choi, B. Galehouse, and C. Yap. Complete subdivision algorithms, II: Isotopic meshing of singular algebraic curves. In *Proc. Int'l Symp. Symbolic and Algebraic Computation (ISSAC'08)*, pages 87–94, 2008. Hagenberg, Austria. Jul 20-23, 2008.

[7] Michael Burr, Sung Woo Choi, Benjamin Galehouse, and Chee K. Yap. Complete subdivision algorithms, II: Isotopic Meshing of Singular Algebraic Curves. In *ISSAC'08:Proc. of the 2008 Int. Symp. on Symbolic and Algebraic Computation*, pages 87–94. ACM press, 2008.

[8] Jin-San Cheng, Xiao-Shan Gao, and Chee K. Yap. Complete numerical isolation of real zeros in general triangular systems. In *Proc. Int'l Symp. Symbolic and Algebraic Comp. (ISSAC'07)*, pages 92–99, 2007. Waterloo, Canada, Jul 29-Aug 1, 2007. DOI: http://doi.acm.org/10.1145/1277548.1277562. In press, Journal of Symbolic Computation.

[9] D. Cox, J. Little, and D. O'Shea. *Using Algebraic Geometry*. Springer, New-York, 1998.

[10] A. Eigenwillig. *Real Root Isolation for Exact and Approximate Polynomials Using Descartes' Rule of Signs*. PhD thesis, Saarland University, Saarbrücken, Germany, 2008.

[11] A. Eigenwillig, M. Kerber, and N. Wolpert. Fast and Exact Geometric Analysis of Real Algebraic Plane Curves. In *ISSAC'07: Proc. of the 2007 Int. Symp. on Symbolic and Algebraic Computation*, pages 151–158. ACM press, 2007.

[12] I. M. Gelfand, M. M. Kapranov, and A. V. Zelevinsky. *Disrcriminants, Resultants and Multidimensional Determinants*. Birkhauser, 1994.

[13] Laureano Gonzalez-Vega and Ioana Necula. Efficient Topology Determination of Implicitly Defined Algebraic Plane Curves. *Comput. Aided Geom. Des.*, 19(9):719–743, 2002.

[14] Irina Voiculescu Huahao Shou, Ralph Martin and et al. Affine arithmetic in matrix form for polynomial evaluation and algebraic curve drawing. *Progress in Natural Science*, 12 (1):77–81, 2002.

[15] L. Kaup and B. Kaup. *Holomorphic Functions of Several Variables*. de Gruyter, 1983.
[16] S. Krishnan, M. Foskey, T. Culver, J. Keyser, and D. Manocha. PRECISE: efficient multiprecision evaluation of algebraic roots and predicates for reliable geometric computation. In *SCG '01: Proc. of the 17th Ann. Symp. on Computational Geometry*, pages 274–283. ACM Press, 2001.
[17] F.S. Macaulay. On some Formula in Elimination. *Proceedings of London Mathematical Society*, pages 3–27, 1902.
[18] R. E. Moore. *Methods and applications of interval analysis*, volume 2 of *SIAM Studies in Applied Mathematics*. SIAM, 1979.
[19] Bernard Mourrain and Jean-Pascal Pavone. Subdivision methods for solving polynomial equations. Technical report, INRIA - Sophia Antipolis, 2005.
[20] A. Neumaier. *Interval Methods for Systems of Equations*. Cambridge University Pres, 1990.
[21] Victor Y. Pan. Sequential and parallel complexity of approximate evaluation of polynomial zeros. *Computers Math. Applic.*, 31(12):97–138, 1996.
[22] S. Petitjean. Algebraic Geometry and Computer Vision: Polynomial Systems, Real and Complex Roots. *J. Math. Imaging Vis.*, 10(3):191–220, 1999.
[23] Michael Sagraloff and Chee K. Yap. An efficient and exact subdivision algorithm for isolating complex roots of a polynomial and its complexity analysis. Draft, unpublished, 2009.
[24] Arnold Schönhage. The fundamental theorem of algebra in terms of computational complexity. *Manuscript, Department of Mathematics, University of Tübingen*, 1982.
[25] B. T. Smith. Error Bounds for Zeros of a Polynomial Based Upon Gerschgorin's Theorems. *J. ACM*, 17(4):661–674, 1970.
[26] A. J. Sommese and C. W. Wampler. *The Numerical Solution of Systems of Polynomials Arising in Engeneering and Science*. World Scientific, Singapore, 2005.
[27] B. Sturmfels. *Solving systems of polynomial equations*, volume 97 of *Regional conference series in mathematics*. AMS, Providence, RI, 2002.
[28] B. Sturmfels and A. Zelevinsky. Multigraded Resultants of Sylvester Type. *J. Algebra*, 163(1):115–127, 1994.
[29] C. K. Yap. *Fundamental Problems in Algorithmic Algebra*. Oxford University Press, 2000.