Clustering Complex Zeros of Triangular Systems of Polynomials*

Remi Imbach[1], Marc Pouget[2], and Chee Yap[1]

[1] Courant Institute of Mathematical Sciences, New York University, USA
remi.imbach@nyu.edu, yap@cs.nyu.edu
[2] Universite de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France
marc.pouget@inria.fr

Abstract. This paper gives the first algorithm for finding a set of natural ϵ-clusters of complex zeros of a regular triangular system of polynomials within a given polybox in $\mathbb{C}^n$, for any given $\epsilon > 0$. Our algorithm is based on a recent near-optimal algorithm of Becker et al (2016) for clustering the complex roots of a univariate polynomial where the coefficients are represented by number oracles.
Our algorithm is based on recursive subdivision. It is local, numeric, certified and handles solutions with multiplicity. Our implementation is compared to with well-known homotopy solvers on various triangular systems. Our solver always gives correct answers, is often faster than the homotopy solvers that often give correct answers, and sometimes faster than the ones that give sometimes correct results.

Keywords: complex root finding · triangular polynomial system · near-optimal root isolation · certified algorithm · complex root isolation · oracle multivariable polynomial · subdivision algorithm · Pellet's theorem.

1 Introduction

This paper considers the fundamental problem of finding the complex solutions of a system $\mathbf{f}(\mathbf{z}) = \mathbf{0}$ of n polynomial equations in n complex variables $\mathbf{z} = (z_1, \ldots, z_n)$. The system $\mathbf{f} = (f_1, \ldots, f_n) : \mathbb{C}^n \to \mathbb{C}^n$ is *triangular* in the sense that $f_i \in \mathbb{C}[z_1, \ldots, z_i]$ for $1 \leq i \leq n$, where $d_{z_i}(f_i) \geq 1$. As in [7], we assume that the system is *regular*: this means that for each i, if $(\alpha_1, \ldots, \alpha_{i-1})$ is a zero of f_{i-1} and $c_i(z_1, \ldots, z_{i-1})$ is the leading coefficient of z_i in f_i, then $c_i(\alpha_1, \ldots, \alpha_{i-1}) \neq 0$. Thus $\mathbf{f}$ is a 0-dimensional system. But unlike [7], we do not assume that the system is square-free: indeed the goal of this paper is to demonstrate new techniques that can properly determine the multiplicity of the root clusters of $\mathbf{f}$, up to any $\epsilon > 0$ resolution.

* Rémi's work is supported by the European Union's Horizon 2020 research and innovation programme No. 676541, NSF Grants # CCF-1563942, # CCF-1564132 and # CCF-1708884. Chee's work is supported by NSF Grants # CCF-1423228 and # CCF-1564132.

Throughout this paper, we use boldface symbols to denote vectors and tuples; for instance $\mathbf{0}$ stands for $(0,\ldots,0)$.

We are interested in finding clusters of solutions of triangular systems and in counting the total multiplicity of solutions in clusters. Solving triangular systems is a fundamental task in polynomial equations solving, since there are many algebraic techniques to decompose the original system into triangular systems.

The problem of isolating the complex solutions of a polynomial system in an initial region-of-interest (ROI) is defined as follows: let $\texttt{Zero}(\mathbf{B},\mathbf{f})$ denote the set of solutions of $\mathbf{f}$ in $\mathbf{B}$, regarded[3] as a multiset.

Local Isolation Problem (LIP):
Given: a polynomial map $\mathbf{f}:\mathbb{C}^n\to\mathbb{C}^n$, a polybox $\mathbf{B}\subset\mathbb{C}^n$, $\epsilon>0$
Output: a set $\{\mathbf{\Delta}^1,\ldots,\mathbf{\Delta}^l\}$ of pairwise disjoint polydiscs of radius $\le\epsilon$ where
- $\texttt{Zero}(\mathbf{B},\mathbf{f})=\bigcup_{j=1}^{l}\texttt{Zero}(\mathbf{\Delta}^j,\mathbf{f})$.
- each $\texttt{Zero}(\mathbf{\Delta}^j,\mathbf{f})$ is a singleton.

This is "local" because we restrict attention to roots in a ROI $\mathbf{B}$. There are two issues with (LIP) as formulated above: deciding if $\texttt{Zero}(\mathbf{\Delta}^j,\mathbf{f})$ is a singleton, and deciding if such a singleton lies in $\mathbf{B}$, are two "zero problems" that require exact computation. Generally, this can only be decided if $\mathbf{f}$ is algebraic. Even in the algebraic case, this may be very expensive. In [27,4] these two issues are side-stepped by defining the local clustering problem which is described next.

Before proceeding, we fix some general notations for this paper. A *polydisc* $\mathbf{\Delta}$ is a vector $(\Delta_1,\ldots,\Delta_n)$ of complex discs. The *center* of $\mathbf{\Delta}$ is the vector of the centers of its components and the *radius* $r(\mathbf{\Delta})$ of $\mathbf{\Delta}$ is the vector of the radii of its components. If δ is any positive real number, we denote by $\delta\mathbf{\Delta}$ the polydisc $(\delta\Delta_1,\ldots,\delta\Delta_n)$ that has the same center as $\mathbf{\Delta}$ and radius $\delta r(\mathbf{\Delta})$. We also say $r(\mathbf{\Delta})\le\delta$ if each component of $r(\mathbf{\Delta})$ is $\le\delta$. A *(square complex) box* B is a complex interval $[\ell_1,u_1]+\boldsymbol{i}([\ell_2,u_2])$ where $u_2-\ell_2=u_1-\ell_1$ and $\boldsymbol{i}:=\sqrt{-1}$; the *width* $w(B)$ of B is $u_1-\ell_1$ and the *center* of B is $u_1+\frac{w(B)}{2}+\boldsymbol{i}(u_2+\frac{w(B)}{2})$. A *polybox* $\mathbf{B}\subseteq\mathbb{C}^n$ is the set $\prod_{i=1}^n B_i$ which is represented by the vector $(B_1,\ldots,B_n)$ of boxes. The *center* of $\mathbf{B}$ is the vector of the centers of its components; the *width* $w(\mathbf{B})$ of $\mathbf{B}$ is the max of the widths of its components. If δ is any positive real number, we denote by $\delta\mathbf{B}$ the polybox $(\delta B_1,\ldots,\delta B_n)$ that has the same center than $\mathbf{B}$ and width $\delta w(\mathbf{B})$. It is also convenient to identify $\mathbf{\Delta}$ as the subset $\prod_{i=1}^n\mathbf{\Delta}_i$ of $\mathbb{C}^n$; a similar remark applies to $\mathbf{B}$.

We introduce three notions to define the local solution clustering problem. Let $\mathbf{a}\in\mathbb{C}^n$ be a solution of $\mathbf{f}(\mathbf{z})=\mathbf{0}$. The *multiplicity* of $\mathbf{a}$ in $\mathbf{f}$, also called the

[3] A *multiset* S is a pair $(\underline{S},\mu)$ where $\underline{S}$ is an ordinary set called the *underlying set* and $\mu:\underline{S}\to\mathbb{N}$ assigns a positive integer $\mu(x)$ to each $x\in\underline{S}$. Call $\mu(x)$ the *multiplicity* of x in S, and $\mu(S):=\sum_{x\in\underline{S}}\mu(x)$ the *total multiplicity* of S. Also, let $|S|$ denote the cardinality of $\underline{S}$. If $|S|=1$, then S is called a *singleton*. We can form the union $S\cup S'$ of two multisets with underlying set $\underline{S}\cup\underline{S'}$, and the multiplicities add up as expected.

intersection multiplicity of $\mathbf{a}$ in $\mathbf{f}$ is classically defined by localization of rings as in [28, Def. 1, p. 61], we denote it by $\#(\mathbf{a}, \mathbf{f})$. An equivalent definition uses dual spaces, see [12, Def. 1, p. 117]. For any set $S \subseteq \mathbb{C}^n$, we denote by $\texttt{Zero}(S, \mathbf{f})$ the multiset of zeros (i.e., solutions) of $\mathbf{f}$ in S, and $\#(S, \mathbf{f})$ the total multiplicity of $\texttt{Zero}(S, \mathbf{f})$. If S is a polydisc, we call $\texttt{Zero}(S, \mathbf{f})$ a *cluster* if it is non-empty, and S is an *isolator* of the cluster. If in addition, we have that $\texttt{Zero}(S, \mathbf{f}) = \texttt{Zero}(3{\cdot}S, \mathbf{f})$, we call $\texttt{Zero}(S, \mathbf{f})$ a *natural cluster* and call S a *natural isolator*. In the context of numerical algorithm, the notion of cluster of solutions is more meaningful than that of solution with multiplicity since the perturbation of a multiple solution generates a cluster. We thus "soften" the problem of isolating the solutions of a triangular system of polynomial equations while counting their multiplicities by translating it into the local solution clustering problem defined as follows:

Local Clustering Problem (LCP):
Given: a polynomial map $\mathbf{f} : \mathbb{C}^n \to \mathbb{C}^n$, a polybox $\mathbf{B} \subset \mathbb{C}^n$, $\epsilon > 0$
Output: a set of pairs $\{(\boldsymbol{\Delta}^1, m^1), \ldots, (\boldsymbol{\Delta}^l, m^l)\}$ where:
- the $\boldsymbol{\Delta}^j$s are pairwise disjoint polydiscs of radius $\leq \epsilon$,
- each $m^j = \#(\boldsymbol{\Delta}^j, \mathbf{f}) = \#(3\boldsymbol{\Delta}^j, \mathbf{f})$
- $\texttt{Zero}(\mathbf{B}, \mathbf{f}) \subseteq \bigcup_{j=1}^{l} \texttt{Zero}(\boldsymbol{\Delta}^j, \mathbf{f}) \subseteq \texttt{Zero}(2\mathbf{B}, \mathbf{f})$.

In this (LCP) reformulation of (LIP), we have removed the two "zero problems" noted above: we output clusters to avoid the first problem, and we allow the output to contain zeroes outside the ROI $\mathbf{B}$ to avoid the second one. We choose $2\mathbf{B}$ for simplicity; it is easy to replace the factor of 2 by $1 + \delta$ for any desired $\delta > 0$.

Overview. In the remaining of this section we explain our contribution, summarize previous work and the local univariate clustering method of [4]. In Sec. 2, we define the notion of *tower of clusters* together with a recursive method to compute the sum of multiplicities of the solutions it contains. Sec. 3 analyzes the loss of precision induced by approximate specialization. Our algorithm for solving the local clustering problem for triangular systems is introduced in Sec. 4. The implementation and experimental results are presented in Sec. 5.

1.1 Our contributions

We propose an algorithm for solving the complex clustering problem for a triangular system $\mathbf{f}(\mathbf{z}) = \mathbf{0}$ with a zero-dimensional solution set. To this end, we propose a formula to count the sum of multiplicities of solutions in a cluster. Our formula is derived from a result of [28] that links the intersection multiplicity of a solution of a triangular system to multiplicities in fibers. We define *towers of clusters* to encode clusters of solutions of a triangular system in stacks (or towers) of clusters of roots of univariate polynomials.

Our algorithm exploits the triangular form of $\mathbf{f} = (f_1, \ldots, f_n)$: the standard idea is to recursively find roots of the form $(\alpha_1, \ldots, \alpha_{n-1})$ of $f_1 = \cdots = f_{n-1} = 0$, then substituting them into f_n to obtain a univariate polynomial $g_n(z_n) =$

$f_n(\alpha_1, \ldots, \alpha_{n-1}, z_n)$. If α_n is a root of $g_n(z_n)$, then we have have extended the solution to $(\alpha_1, \ldots, \alpha_n)$ of the original $\mathbf{f}$. The challenge is to extend this idea to compute clusters of zeros of $\mathbf{f}$ from clusters of zeros of $f_1 = \cdots = f_{n-1} = 0$. Moreover, we want to allow the coefficients of each f_i to be oracle numbers. The use of oracle numbers allows us to treat polynomial systems whose coefficients are algebraic numbers and beyond.

To compute clusters of roots of a univariate polynomial given as an oracle, we rely on the recent algorithm described in [4], based on a predicate introduced in [5] that combines Pellet's theorem and Graeffe iterations to determine the number of roots counted with multiplicities in a complex disc; this predicate is called *soft* because it only requires the polynomial to be known as approximations. It is used in a subdivision framework combined with Newton iterations to achieve a near optimal complexity.

We implemented our algorithm and made it available as the `Julia`[4] package `Ccluster.jl`[5]. Our experiments show that it advantageously compares to major homotopy solvers for solving random dense triangular systems in terms of solving times and reliability (*i.e.* getting the correct number of solutions and the correct multiplicity structures). Homotopy solving is more general because it deals with any polynomial system. We also propose experiments with triangular systems obtained with elimination procedures.

1.2 Related work

There is a vast literature on solving polynomial systems and we can only refer to book surveys and references therein, see for instance [13,25]. On the algebraic side, symbolic tools like Groebner basis, resultant, rational univariate parametrization or triangularization, find an equivalent triangular system or set thus reducing the problem to the univariate case. Being symbolic, these methods handle all input, in particular with solutions with multiplicities, and are certified but at the price of a high complexity that limits their use in practice. Implementations of hybrid symbolic-numeric solvers are available for instance in `Singular`[6] via `solve.lib` or in `Maple` via `RootFinding[Isolate]`.

On the numerical side, one can find subdivision and homotopy methods. The main advantage of subdivision methods is their locality: the practical complexity depends on the size of the solving domain and the number of solutions in this domain. Their main drawback is that they are only practical for low dimensional systems. On the other hand, homotopy methods are efficient for high dimensional systems, they are not local but solutions are computed independently from one another. Numerical methods only work for restricted classes of systems and the certification of the output remains a challenge. Multiprecision arithmetic, interval analysis, deflation and α-theory are now classical tools to address this certification issue [15,22,6,26].

[4] https://julialang.org/
[5] https://github.com/rimbach/Ccluster.jl
[6] https://www.singular.uni-kl.de/

In the univariate case, practical certified algorithms are now available for real and complex solving that match the best known complexity bounds together with efficient implementations [19,17]. For the bivariate case, the problem of solving a triangular system can be seen as a univariate isolation in an extension field. The most recent contributions in this direction presenting algorithms together with complexity analysis are [23,24].

Only a few work address the specific problem of solving triangular polynomial systems. The solving can then be performed coordinate by coordinate by specialization and univariate solving in fibers. When the systems only have regular solutions, extensions of classical univariate isolation algorithms to polynomial with interval coefficients have been proposed [9,14,7]. In the presence of multiple solutions, one approach is to use a symbolic preprocessing to further decompose the system in regular sub-systems. Another approach is the sleeve method with separation bounds [8]. The authors of [28] propose a formula to compute the multiplicity of a solution of a triangular system: the latter multiplicity is the product of the multiplicities of the components of a solution in the fibers. Then, by using square free factorization of univariate polynomials specialized in fibers, they describe an algorithm to retrieve the real solutions of a triangular system with their multiplicities. In [20], the method of Local Generic Position is adapted to the special case of triangular systems with the advantage of only using resultant computations (instead of Goebner basis), multiplicities are also computed.

1.3 Definitions and Notation

Convention for Vectors. We introduce some general conventions for vectors that will simplify the following development. Vectors are indicated by bold fonts. If $\boldsymbol{v} = (v_1, \ldots, v_n)$ is an n-vector, and $i = 1, \ldots, n$, then the i-th component v_i is[7] denoted $\boldsymbol{v}_i$ and the i-vector $(v_1, \ldots, v_i)$ is denoted $\boldsymbol{v}_{(i)}$. Thus $\boldsymbol{v} = (\boldsymbol{v}_{(n-1)}, \boldsymbol{v}_n)$, and "$\boldsymbol{v} = \boldsymbol{v}_{(n)}$" is an idiomatic way of saying that $\boldsymbol{v}$ is an n-vector. Because of the subscript convention, we will superscripts such as $\boldsymbol{v}^1, \boldsymbol{v}^2$, etc, to distinguish among a set of related n-vectors.

Normed Vector Spaces. In order to do error analysis, we need to treat $\mathbb{C}[\boldsymbol{z}]$ and $\mathbb{C}^n$ as normed vector spaces: for $f \in \mathbb{C}[\boldsymbol{z}]$ and $\boldsymbol{b} \in \mathbb{C}^n$, let $\|f\|$ and $\|\boldsymbol{b}\|$ denote the infinity norm on polynomials and vectors, respectively. We use the following *perturbation convention*: let $\delta \geq 0$. Then we will write $f \pm \delta$ to denote some polynomial $\widetilde{f} \in \mathbb{C}[\boldsymbol{z}]$ that satisfies $\|f - \widetilde{f}\| \leq \delta$. Similarly, $\boldsymbol{b} \pm \delta$ denotes some vector $\widetilde{\boldsymbol{b}} \in \mathbb{C}^n$ that satisfies $\|\boldsymbol{b} - \widetilde{\boldsymbol{b}}\| \leq \delta$. If $\delta \leq 2^{-L}$ then $\widetilde{\boldsymbol{b}}$ and $\widetilde{\boldsymbol{f}}$ are called *L-bit approximations* of $\boldsymbol{b}$ and $\boldsymbol{f}$, respectively.

We define the *degree sequence* of $f \in \mathbb{C}[\boldsymbol{z}]$ to be $\boldsymbol{d} = \boldsymbol{d}(f)$ where $\boldsymbol{d}_i$ is the degree of z_i in f. If $\boldsymbol{b} \in \mathbb{C}^k$ $(k = 1, \ldots, n)$, let $f(\boldsymbol{b})$ denote the polynomial that results from the substitution $\boldsymbol{z}_i \to \boldsymbol{b}_i$ (for $i = 1, \ldots, k$). The result is a polynomial $f(\boldsymbol{b}) \in \mathbb{C}[z_{k+1}, \ldots, z_n]$ called the *specialization* of f by $\boldsymbol{b}$. Note that

[7] In general, $\boldsymbol{v}_i \neq v_i$ since $\boldsymbol{v}$ and v_i are independent variables. So our bold font variables $\boldsymbol{v}$ do not entail the existence of non-bold font counterparts such as v_i.

$f(\boldsymbol{b})$ is a polynomial in at most $n-k$ variables. In particular, when $n=k$, then $f(\boldsymbol{b})$ is a constant (called the *evaluation* of f at $\boldsymbol{b}$). For instance, suppose $\boldsymbol{b} \in \mathbb{C}^n$, then $f(\boldsymbol{b}_{(n-1)})$ is a polynomial in z_n and $f(\boldsymbol{b}_{(n-1)})(\boldsymbol{b}_n) = f(\boldsymbol{b})$.

If $B \subseteq \mathbb{C}$ is a box with center c and width w, we denote by $\Delta(B)$ the disc with center c and radius $\frac{3}{4}w$. Note that $\Delta(B)$ contains B. If $\mathbf{B} \subset \mathbb{C}^n$ is a polybox, let $\Delta(\mathbf{B})$ be the polydisc where $\Delta(\mathbf{B})_i = \Delta(\mathbf{B}_i)$.

Oracle Computational Model. We use two kinds of numbers in our algorithms: an explicit kind which is standard in computing, and an implicit kind which we call "oracles". Our explicit numbers are dyadic numbers (i.e., bigFloats), $\mathbb{D} := \{n2^m : n, m \in \mathbb{Z}\}$. A pair (n, m) of integers represents the *nominal value* of $n2^m \in \mathbb{D}$. However, we also want this pair to represent the interval $[(n-\frac{1}{2})2^m, (n+\frac{1}{2})2^m]$. To distinguish between them, we write $(n, m)_0$ for the nominal value, and $(n, m)_1$ for the interval of width 2^m. Call $(n, m)_1$ an *L-bit dyadic interval* if $m \leq -L$ (so the interval has width at most 2^{-L}). Note that $(2n, m)_1$ and $(n, m+1)_1$ are different despite having the same nominal value. As another example, note that $(0, m)_1$ is the interval $[-2^{m-1}, 2^{m-1}]$. When we say a box, disc, polybox, etc, is *dyadic*, it means that all its parameters are given by dyadic numbers. The set of closed intervals with dyadic endpoints is denoted $\square\mathbb{D}$. Also, let $\square^n\mathbb{D}$ denote n-dimensional dyadic boxes.

The implicit numbers in our algorithms are functions: for any real number $x \in \mathbb{R}$, an *oracle for x* is a function $\mathcal{O} : \mathbb{Z} \to \square\mathbb{D}$ such that $\mathcal{O}_x(L)$ is an L-bit dyadic interval containing x. There is normally no confusion in identifying the real number x with *any* oracle function $\mathcal{O}_x$ for x. Moreover, we write $(x)_L$ instead of $\mathcal{O}_x(L)$. E.g., if x is a real algebraic number with defining polynomial $p \in \mathbb{Z}[X]$ and isolating interval I, we may define an oracle $\mathcal{O}_x = \mathcal{O}(p, I)$ for x in a fairly standard way. Next, an oracle $\mathcal{O}_z$ for a complex number $z = x + \boldsymbol{i}y$ is a function $\mathcal{O}_z : \mathbb{Z} \to \square^2\mathbb{D}$ such that $\mathcal{O}_z(L) = \mathcal{O}_x(L) + \boldsymbol{i}\mathcal{O}_y(L)$ where $\mathcal{O}_x, \mathcal{O}_y$ are oracles for x and y. Again, we may identify z with any oracle $\mathcal{O}_z$, and write $(z)_L$ instead of $\mathcal{O}_z(L)$. For polynomials $f \in \mathbb{C}[\boldsymbol{z}_{(n)}]$ in $n \geq 2$ variables, we assume a sparse representation, $f = \sum_{\boldsymbol{\alpha} \in Supp(f)} f_{\boldsymbol{\alpha}} \boldsymbol{z}^{\boldsymbol{\alpha}}$ with fixed support $Supp(f) \subseteq \mathbb{N}^n$, with coefficients $f_{\boldsymbol{\alpha}} \in \mathbb{C} \setminus \{0\}$, and $\boldsymbol{z}^{\boldsymbol{\alpha}} := \prod_{i=1}^n \boldsymbol{z}_i^{\boldsymbol{\alpha}_i}$ are power products. An oracle $\mathcal{O}_f$ for f amounts to having oracles for each coefficient $f_{\boldsymbol{\alpha}}$ of f. Moreover $\mathcal{O}_f(L)$ may be written $(f)_L$ is the interval polynomial whose coefficients are $(f_\alpha)_L$. Call $(f)_L$ a *dyadic interval polynomial*.

1.4 Oracles for Root Cluster of Univariate Polynomials

The starting point for this paper is the fundamental result that the Local Clustering Problem (LCP) has been solved in the univariate setting:

Proposition 1 (See [4,5]) *There is an algorithm $Cluster(f, B, \epsilon)$ that solves the Local Clustering Problem when $f : \mathbb{C} \to \mathbb{C}$ is a univariate oracle polynomial.*

In other words, the output of $Cluster(f, B, \epsilon)$ is a set $\{(\Delta_i, m_i) : i = 1, \ldots, k\}$ such that each Δ_i is a natural ϵ-isolator, and

$$\texttt{Zero}(B, f) \subseteq \bigcup_{i=1}^{k} \texttt{Zero}(\Delta_i, f) \subseteq \texttt{Zero}(2B, f).$$

To make this result the basis of our multivariate clustering algorithm, we need to generalize this result. In particular, we need to be able to further refine each output (Δ_i, m_i) of this algorithm. If (Δ_i, m_i) represents the cluster C_i of roots, we want to get better approximation of C_i, i.e., we want to treat C_i like number oracles. Fortunately, the algorithm in [4,5] already contains the tools to do this. What is lacking is a conceptual framework to capture this.

Our goal is to extend the concept of number oracles to "cluster oracles". To support the several modifications which are needed, we revise our previous view of "oracles as functions". We now think of an oracle $\mathcal{O}$ as a computational object with *state information*, and which can transform itself in order to update its state information. For any $L \in \mathbb{Z}$, recall that $\mathcal{O}(L)$ is a dyadic object that is at least L-bit accurate. E.g., if $\mathcal{O}$ is the oracle for $x \in \mathbb{R}$, $\mathcal{O}(L)$ is an interval containing x of width $\leq 2^{-L}$. But now, we say that oracle is transformed to a new oracle which we shall denote by "$(\mathcal{O})_L$" whose state information is $\mathcal{O}(L)$. In general, let $\sigma(\mathcal{O})$ denote the state information in $\mathcal{O}$. Next, for a cluster $C \subseteq \mathbb{C}$ of roots of a univariable polynomial $p(z) \in \mathbb{C}[z]$, its oracle $\mathcal{O}_C$ has state $\sigma(\mathcal{O}_C)$ that is a pair (Δ, m) where $\Delta \subseteq \mathbb{C}$ is a dyadic disc satisfying $C = \texttt{Zero}(\Delta, p) = \texttt{Zero}(3\Delta, p)$ and m is the total multiplicity of C. Thus C is automatically a natural cluster. We say $\mathcal{O}_C$ is *L-bit accurate* if the radius of Δ is at most 2^{-L}. Intuitively, we expect $(\mathcal{O}_C)_L$ to be an oracle for C that is L-bit accurate. Unfortunately, this may be impossible unless C is a singleton cluster. In general, we may have to split C into two or more clusters. We therefore need one more extension: the map $L \mapsto (\mathcal{O}_C)_L$ returns a set

$$\{\mathcal{O}_{C_1}, \ldots, \mathcal{O}_{C_k}\}, \quad \text{(for some } k \geq 1)$$

of cluster oracles with the property that $C = \cup_{i=1}^{k} C_i$ (union of multisets), and each $\mathcal{O}_{C_i}$ is L-bit accurate. We generalize Proposition 1 so that it outputs a collection of cluster oracles:

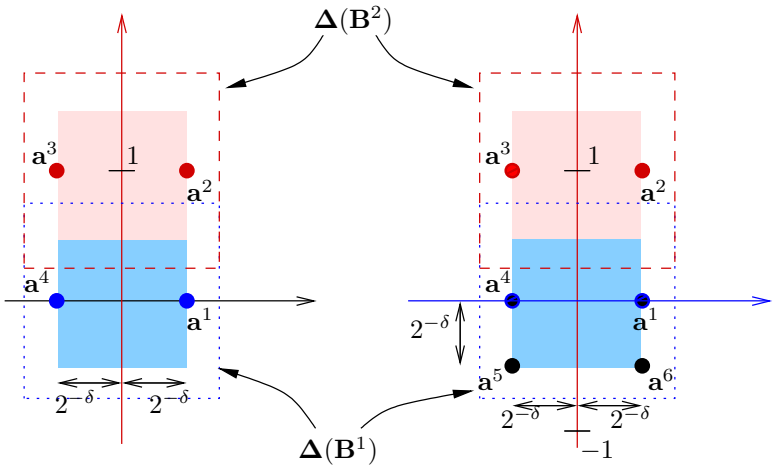
Proposition 2 (See [4,5,17]) *Let $\mathcal{O}_f$ be an oracle for a univariate polynomial $f : \mathbb{C} \to \mathbb{C}$. There is an algorithm $ClusterOracle(\mathcal{O}_f, B, L)$ that returns a set $\{\mathcal{O}_{C_i} : i = 1, \ldots, k\}$ of cluster oracles such that*

$$\texttt{Zero}(B, f) \subseteq \bigcup_{i=1}^{k} C_i \subseteq \texttt{Zero}(2B, f).$$

and each $\mathcal{O}_{C_i}$ is L-bit accurate.

2 Sum of multiplicities in clusters of solutions

We extend in Sec. 2.2 a result of [28] to an inductive formula giving the sum of multiplicities of solutions of a triangular system in a cluster. In Sec. 2.3, we



$\#(\mathbf{a}^3, \mathbf{g}) = 1 \times 1$	$\#(\mathbf{a}^2, \mathbf{g}) = 1 \times 1$	$\#(\mathbf{a}^3, \mathbf{h}) = 1 \times 1$	$\#(\mathbf{a}^2, \mathbf{h}) = 2 \times 1$
$\#(\mathbf{a}^4, \mathbf{g}) = 1 \times 1$	$\#(\mathbf{a}^1, \mathbf{g}) = 1 \times 1$	$\#(\mathbf{a}^4, \mathbf{h}) = 1 \times 1$	$\#(\mathbf{a}^1, \mathbf{h}) = 2 \times 1$
$\#(\boldsymbol{\Delta}(\mathbf{B}^1), \mathbf{g}) = 2 \times 1$		$\#(\mathbf{a}^5, \mathbf{h}) = 1 \times 2$	$\#(\mathbf{a}^6, \mathbf{h}) = 2 \times 2$
$\#(\boldsymbol{\Delta}(\mathbf{B}^2), \mathbf{g}) = 2 \times 1$		$\#(\boldsymbol{\Delta}(\mathbf{B}^1), \mathbf{h}) = 3 \times 3$	
		$\#(\boldsymbol{\Delta}(\mathbf{B}^2), \mathbf{h}) = 3 \times 1$	

Fig. 1. On the left (resp. right), the solutions of $\mathbf{g}(\mathbf{z}) = 0$ (resp. $\mathbf{h}(\mathbf{z}) = 0$) defined in Eq. 1 (resp. Eq. 2) with $\delta = 1$. $\mathbf{B}^1$ (resp. $\mathbf{B}^2$) is the polybox of $\mathbb{C}^2$ with center $(0, 0)$ (resp. $(0, 1)$) and width $2 * 2^{-\delta}$. The boxes in dashed lines are the real parts of $\boldsymbol{\Delta}(\mathbf{B}^1)$ and $\boldsymbol{\Delta}(\mathbf{B}^2)$. In the frame, the multiplicities of solutions of each system are computed with the formula of Zhang (Proposition 3) and Thm. 1.

introduce a representation of clusters of solutions of $\mathbf{f}$ called *tower representation*, reflecting the triangular form of $\mathbf{f}$. Sec. 2.1 presents two illustrative examples.

2.1 Two examples

Let $\delta > 0$ be an integer. We define the systems $\mathbf{g}(\mathbf{z}) = (g_1(z_1), g_2(z_1, z_2)) = \mathbf{0}$ and $\mathbf{h}(\mathbf{z}) = (h_1(z_1), h_2(z_1, z_2)) = \mathbf{0}$ as follows:

$$(\mathbf{g}(\mathbf{z}) = \mathbf{0}) : \begin{cases} (z_1 - 2^{-\delta})(z_1 + 2^{-\delta}) = 0 \\ (z_2 - 2^{2\delta} z_1^2) z_2 = 0 \end{cases} \tag{1}$$

$$(\mathbf{h}(\mathbf{z}) = \mathbf{0}) : \begin{cases} (z_1 - 2^{-\delta})^2 (z_1 + 2^{-\delta}) = 0 \\ (z_2 + 2^{\delta} z_1^2)^2 (z_2 - 1) z_2 = 0 \end{cases} \tag{2}$$

$\mathbf{g}(\mathbf{z}) = 0$ has 4 solutions: $\mathbf{a}^1 = (2^{-\delta}, 0)$, $\mathbf{a}^2 = (2^{-\delta}, 1)$, $\mathbf{a}^3 = (-2^{-\delta}, 1)$ and $\mathbf{a}^4 = (-2^{-\delta}, 0)$. $\mathbf{h}(\mathbf{z}) = 0$ has 6 solutions: $\mathbf{a}^1$, $\mathbf{a}^2$, $\mathbf{a}^3$, $\mathbf{a}^4$, $\mathbf{a}^5 = (-2^{-\delta}, -2^{-\delta})$ and $\mathbf{a}^6 = (2^{-\delta}, -2^{-\delta})$. For $1 \leq i \leq 6$, let $\mathbf{a}^i = (a_1^i, a_2^i)$. The solutions of both $\mathbf{g} = 0$ and $\mathbf{h} = 0$ are depicted in Fig. 1.

2.2 Sum of multiplicities in a cluster

We recall a theorem of Zhang [28] for counting multiplicities of solutions of triangular systems, based multiplicities in fibers. We may rephrase it inductively:

Proposition 3 ([28]) *Let $n \geq 2$ and $\mathbf{a} \in \mathbb{C}^n$ be a solution of the triangular system $\mathbf{f}(\mathbf{z}) = 0$. The multiplicity of $\mathbf{a}$ in $\mathbf{f}$ is*

$$\#(\mathbf{a}, \mathbf{f}) = \#(\mathbf{a}_n, \mathbf{f}_n(\mathbf{a}_{(n-1)})) \times \#(\mathbf{a}_{(n-1)}, \mathbf{f}_{(n-1)}).$$

We extend Proposition 3 to a formula giving the total multiplicity of a cluster $\texttt{Zero}(\boldsymbol{\Delta}, \mathbf{f})$.

Theorem 1. *Let $\texttt{Zero}(\boldsymbol{\Delta}, \mathbf{f})$ be a cluster of solutions of the triangular system $\mathbf{f}(\mathbf{z}) = 0$. If there is an integer $m \geq 1$ so that for any solution $\mathbf{a} \in \texttt{Zero}(\boldsymbol{\Delta}, \mathbf{f})$, one has $m = \#(\boldsymbol{\Delta}_n, \mathbf{f}_n(\mathbf{a}_{(n-1)}))$, then*

$$\#(\boldsymbol{\Delta}, \mathbf{f}) = m \times \#(\boldsymbol{\Delta}_{(n-1)}, \mathbf{f}_{(n-1)}).$$

where $\#(\boldsymbol{\Delta}_{(n-1)}, \mathbf{f}_{(n-1)}) = 1$ when $n = 1$.

Let us apply Proposition 3 to compute the multiplicities of solutions of $\mathbf{g}(\mathbf{z}) = 0$ and $\mathbf{h}(\mathbf{z}) = 0$ (see Eq. 1 and Eq. 2). $\mathbf{a}^1$ has multiplicity 1 in $\mathbf{g}$: $\#(\mathbf{a}^1, \mathbf{g}) = \#(a_1^1, g_1) \times \#(a_2^1, g_2(a_1^1)) = 1 \times 1$. $\mathbf{a}^1$ has multiplicity 2 in $\mathbf{h}$: $\#(\mathbf{a}^1, \mathbf{h}) = \#(a_1^1, h_1) \times \#(a_2^1, h_2(a_1^1)) = 2 \times 1$. The multiplicities of other solutions are given in fig. 1.

Let $\mathbf{B}^1 = (B_1^1, B_2^1)$ be the polybox centered in $(0, 0)$ having width $2 \times 2^{-\delta}$. $\texttt{Zero}(\boldsymbol{\Delta}(\mathbf{B}^1), \mathbf{g}) = \{\mathbf{a}^1, \mathbf{a}^4\}$ and $\#(\boldsymbol{\Delta}(\mathbf{B}^1), \mathbf{g}) = 2$. Since $\#(\Delta(B_2^1), \mathbf{g}_n(\mathbf{a^1}_{(n-1)})) = \#(\Delta(B_2^1), \mathbf{g}_n(\mathbf{a^4}_{(n-1)})) = 1$, applying Thm. 1 yields $\#(\boldsymbol{\Delta}(\mathbf{B}^1), \mathbf{g}) = 2 \times 1$.

$\texttt{Zero}(\boldsymbol{\Delta}(\mathbf{B}^1), \mathbf{h}) = \{\mathbf{a}^1, \mathbf{a}^4, \mathbf{a}^5, \mathbf{a}^6\}$ and $\#(\boldsymbol{\Delta}(\mathbf{B}^1), \mathbf{h}) = 9$. Again, one has $\#(\Delta(B_2^1), \mathbf{h}_n(\mathbf{a^1}_{(n-1)})) = \#(\Delta(B_2^1), \mathbf{h}_n(\mathbf{a^4}_{(n-1)})) = 3$. Thus applying Thm. 1 yields $\#(\boldsymbol{\Delta}(\mathbf{B}^1), \mathbf{h}) = 3 \times 3$.

Let $\mathbf{B}^2$ be the polybox centered in $(0, 1)$ having width $2 \times 2^{-\delta}$. One can apply Thm. 1 to obtain $\#(\boldsymbol{\Delta}(\mathbf{B}^2), \mathbf{g}) = 2 \times 1$ and $\#(\boldsymbol{\Delta}(\mathbf{B}^2), \mathbf{h}) = 3 \times 1$. The real parts of $\boldsymbol{\Delta}(\mathbf{B}^1)$ and $\boldsymbol{\Delta}(\mathbf{B}^2)$ are depicted in Fig. 1.

Proof (of Thm. 1.). Remark that $\texttt{Zero}(\boldsymbol{\Delta}, \mathbf{f}) = \{\mathbf{a} \in \boldsymbol{\Delta} | \mathbf{f}(\mathbf{a}) = \mathbf{0}\}$ can be defined in an inductive way as $\texttt{Zero}(\boldsymbol{\Delta}, \mathbf{f}) = \{(\mathbf{b}, c) \in \boldsymbol{\Delta} | \mathbf{b} \in \texttt{Zero}(\boldsymbol{\Delta}_{(n-1)}, \mathbf{f}_{(n-1)})$ and $c \in \texttt{Zero}(\boldsymbol{\Delta}_n, \mathbf{f}_n(\mathbf{b}))\}$. Using Proposition 3, we may write

$$\begin{aligned}
\#(\boldsymbol{\Delta}, \mathbf{f}) &= \sum_{(\mathbf{b},c) \in \texttt{Zero}(\boldsymbol{\Delta}, \mathbf{f})} \#(\mathbf{b}, \mathbf{f}_{(n-1)}) \times \#(c, \mathbf{f}_n(\mathbf{b})) \\
&= \sum_{\mathbf{b} \in \texttt{Zero}(\boldsymbol{\Delta}_{(n-1)}, \mathbf{f}_{(n-1)})} \left(\#(\mathbf{b}, \mathbf{f}_{(n-1)}) \times \sum_{c \in \texttt{Zero}(\boldsymbol{\Delta}_n, \mathbf{f}_n(\mathbf{b}))} \#(c, \mathbf{f}_n(\mathbf{b})) \right) \\
&= \sum_{\mathbf{b} \in \texttt{Zero}(\boldsymbol{\Delta}_{(n-1)}, \mathbf{f}_{(n-1)})} \#(\mathbf{b}, \mathbf{f}_{(n-1)}) \times m \\
&= m \times \#(\boldsymbol{\Delta}_{(n-1)}, \mathbf{f}_{(n-1)}).
\end{aligned}$$

□

2.3 Tower Representation

Definition 1 (Tower Representations). *Let* $\mathbf{\Delta} \subseteq \mathbb{C}^n$ *be a polydisc and* $\mathbf{m}$ *a* n*-vector of positive integers. The pair* $(\mathbf{\Delta}, \mathbf{m})$ *is a* tower *(relative to* $\mathbf{f}$*) if it satisfies: if* $n = 1$*, then* $(\mathbf{\Delta}, \mathbf{m}) = (\mathbf{\Delta}_1, \mathbf{m}_1)$ *is a cluster representation relative to* $\mathbf{f}$*. Inductively, if* $n > 1$ *then:*

(i) $(\mathbf{\Delta}_{(n-1)}, \mathbf{m}_{(n-1)})$ *is a tower relative to* $\mathbf{f}_{(n-1)}$
(ii) $\forall \mathbf{b} \in \mathbf{\Delta}_{(n-1)}$, $(\mathbf{\Delta}_n, \mathbf{m}_n)$ *is a cluster representation relative to* $\mathbf{f}_n(\mathbf{b})$.

The height *of the tower* $(\mathbf{\Delta}, \mathbf{m})$ *is* n.

If we replace 'cluster' by 'natural cluster' in the above definition, then $(\mathbf{\Delta}, \mathbf{m})$ a *natural tower*. If $\mathbf{B}$ is a polybox, and $(\Delta(\mathbf{B}), \mathbf{m})$ is a tower relative to $\mathbf{f}$, then we can also call $(\mathbf{B}, \mathbf{m})$ a *(polybox) tower* relative to $\mathbf{f}$. Below, we will only consider natural towers and will omit the word natural.

Let $\mathbf{g}, \mathbf{h}$ be defined as in Eqs. 1 and 2 and $\mathbf{B}^1 = (B_1^1, B_2^1)$, $\mathbf{B}^2 = (B_1^2, B_2^2)$ be as defined in 2.2. The pair $(\Delta(B_1^1), 3)$ is a tower relative to h_1. Moreover, if $\delta \geq 3$, $(\Delta(\mathbf{B}^1), (3, 3))$ and $(\Delta(\mathbf{B}^2), (3, 1))$ are towers relative to $\mathbf{h}$. Consider the polynomial $h_2(z_1, z_2) = (z_2 + 2^\delta z_1^2)^2(z_2 - 1)z_2$. If $z_2 \in 3\Delta(B_2^1)$ then $|z_2| < \frac{3\times 3}{16} < 1$ and for any $z_1 \in \Delta(B_1^1)$, h_2 has 3 roots counted with multiplicity in $\Delta(B_2^1)$ and in $3\Delta(B_2^1)$. Hence for any $b \in \Delta(B_1^1)$, $(\Delta(B_2^1), 3)$ is a tower relative to $h_2(b)$ then $(\Delta(\mathbf{B}^1), (3, 3))$ is a tower relative to $\mathbf{h}$. Similarly, $(\Delta(\mathbf{B}^2), (3, 1))$ is a tower relative to $\mathbf{h}$. $(\mathbf{B}^1, (3, 3))$ and $(\mathbf{B}^2, (3, 1))$ are (polybox) towers relative to $\mathbf{h}$.

In contrast, although $(\Delta(B_1^1), 2)$ is a tower relative to g_1, there exist no tower relative to $\mathbf{g}$ having $\mathbf{B}^1$ or $\mathbf{B}^2$ as box: $-2^{-\delta}$, 0 and $2^{-\delta}$ are three points of $B_1^1 = B_1^2$; consider the three polynomials $g_2(-2^{-\delta})$, $g_2(0)$ and $g_2(2^{-\delta})$. $g_2(-2^{-\delta})$ and $g_2(2^{-\delta})$ have each 1 root of multiplicity 1 in B_2^1 while $g_2(0)$ has 1 root of multiplicity 2 in B_2^1: there is no $\mathbf{m}$ that satisfy condition *(ii)* of Def. 1. In the case of $\mathbf{B}^2$, $g_2(-2^{-\delta})$ and $g_2(2^{-\delta})$ have both 1 root of multiplicity 1 in B_2^2 while $g_2(0)$ has no root in B_2^2.

An immediate consequence of the previous theorem is

Corollary 4 *Let* $(\mathbf{\Delta}, \mathbf{m})$ *be a tower relative to* $\mathbf{f}$ *of height* $n > 1$*. Then*

$$\#(\mathbf{\Delta}, \mathbf{f}) = \mathbf{m}_n \times \#(\mathbf{\Delta}_{(n-1)}, \mathbf{f}_{(n-1)}).$$

Inductively, we have $\#(\mathbf{\Delta}, \mathbf{f}) = \prod_{i=1}^n \mathbf{m}_i$

Remark finally that if $(\mathbf{B}, \mathbf{m})$ is a tower relative to $\mathbf{f}_{(n-1)}$ and f is an oracle for $\mathbf{f}_n(\mathbf{b})$ for any $\mathbf{b} \in \Delta(\mathbf{B})$, one can use *Cluster*, as specified in Prop. 1, to compute clusters of $\mathbf{f}_n(\mathbf{b})$ for any $\mathbf{b} \in \Delta(\mathbf{B})$ in a box B. If this returns a list $\{(B^j, m^j) | 1 \leq j \leq l\}$, then for all $1 \leq j \leq l$, $((\Delta(\mathbf{B}), \Delta(B^j)), (\mathbf{m}, m^j))$ is a tower relative to $\mathbf{f}$, and from corollary 4, $\#((\Delta(\mathbf{B}), \Delta(B^j)), \mathbf{f}) = m^j \times \prod_{k=1}^{n-1} \mathbf{m}_k$. Moreover, $\texttt{Zero}((\mathbf{B}, B), \mathbf{f}) \subseteq \bigcup_{j=1}^l \texttt{Zero}((\Delta(\mathbf{B}), \Delta(B^j)), \mathbf{f}) \subseteq \texttt{Zero}((2\mathbf{B}, 2B), \mathbf{f})$. In other words, $\{((\Delta(\mathbf{B}), \Delta(B^j)), m^j \times \prod_{k=1}^{n-1} \mathbf{m}_k) | 1 \leq j \leq l\}$ is a solution for the clustering problem in $(\mathbf{B}, B)$.

We show in Sec. 4 how to setup an oracle for $\mathbf{f}_n(\mathbf{b})$ for any $\mathbf{b} \in \Delta(\mathbf{B})$. This oracle may refine $(\mathbf{B}, \mathbf{m})$ and split it into several clusters.

3 Error Analysis of Approximate Specializations

The proofs for this section is found in the Appendix.

Given $f, \widetilde{f} \in \mathbb{C}[\boldsymbol{z}] = \mathbb{C}[\boldsymbol{z}_{(n)}]$ and $\boldsymbol{b}, \widetilde{\boldsymbol{b}} \in \mathbb{C}^n$, our basic goal is to bound the evaluation error

$$\|f(\boldsymbol{b}) - \widetilde{f}(\widetilde{\boldsymbol{b}})\|$$

in terms of $\delta_f := \|f - \widetilde{f}\|$ and $\delta_{\boldsymbol{b}} := \|\boldsymbol{b} - \widetilde{\boldsymbol{b}}\|$. This will be done by induction on n. Our analysis aims not just to produce some error bound, but to express this error in terms that are easily understood, and which reveals the underlying inductive structure. Towards this end, we introduce the following *β-bound* function: if d is a positive integer and $b \in \mathbb{C}$,

$$\beta(d, b) := \sum_{i=0}^{d} |b|^i. \tag{3}$$

Let $\boldsymbol{d} = \boldsymbol{d}(f)$, i.e., $\boldsymbol{d}_i = deg_{\boldsymbol{z}_i}(f)$ for each i. The *support* of f is $Supp(f) \subseteq \mathbb{N}^n$ where $f = \sum_{\boldsymbol{\alpha} \in Supp(f)} c_{\boldsymbol{\alpha}} \boldsymbol{z}^{\boldsymbol{\alpha}}$ where $c_{\boldsymbol{\alpha}} \in \mathbb{C} \setminus \{0\}$. Here, $\boldsymbol{z}^{\boldsymbol{\alpha}} := \prod_{i=1}^{n} \boldsymbol{z}_i^{\boldsymbol{\alpha}_i}$. We assume that $Supp(\widetilde{f}) \subseteq Supp(f)$. Our induction variable is $k = 1, \ldots, n$. For $\boldsymbol{\alpha} \in \mathbb{N}^n$, let $\pi_k(\boldsymbol{\alpha}) := (0, \ldots, 0, \boldsymbol{\alpha}_{k+1}, \ldots, \boldsymbol{\alpha}_n)$. E.g., if $k = n$ then $\pi_k(\boldsymbol{\alpha}) = \boldsymbol{0}$. Thus $\boldsymbol{\alpha} - \pi_k(\boldsymbol{\alpha}) = (\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_k, 0, \ldots, 0)$. Next define $Supp_k(f) := \{\pi_k(\boldsymbol{\alpha}) : \boldsymbol{\alpha} \in Supp(f)\}$. With this notation, we can write

$$f = \sum_{\boldsymbol{\alpha} \in Supp_k(f)} f_{\boldsymbol{\alpha}} \boldsymbol{z}^{\boldsymbol{\alpha}} \tag{4}$$

where each $f_{\boldsymbol{\alpha}} \in \mathbb{C}[\boldsymbol{z}_{(k)}]$. E.g., if $k = n$ then $Supp_k(f) = \{\boldsymbol{0}\}$ and so $f_{\boldsymbol{0}} = f$. Assume that we are given $f, \widetilde{f} \in \mathbb{C}[\boldsymbol{z}] = \mathbb{C}[\boldsymbol{z}_{(n)}]$ and $\boldsymbol{b}, \widetilde{\boldsymbol{b}} \in \mathbb{C}$. Also the degree sequences satisfies $\boldsymbol{d}(\widetilde{f}) \leq \boldsymbol{d}(f)$, that is the inequality holds componentwise. Then we may define these quantities for $k = 1, \ldots, n$:

$$\begin{array}{rcll} \delta_k \boldsymbol{b} & := & |\boldsymbol{b}_k - \widetilde{\boldsymbol{b}}_k|, & \\ \delta_k f & := & \|f(\boldsymbol{b}_{(k)}) - \widetilde{f}(\widetilde{\boldsymbol{b}}_{(k)})\| & (\text{with } \delta_0 f = \|f - \widetilde{f}\|), \\ \boldsymbol{\beta}_k & := & \beta(\boldsymbol{d}_k, \boldsymbol{b}_k) & \\ \widetilde{\boldsymbol{\beta}}_k & := & \beta(\boldsymbol{d}_k, |\boldsymbol{b}_k| + \delta_k \boldsymbol{b}). & \end{array}$$

Note that δ_k is a operator that must attach to some function f or vector $\boldsymbol{b}$ to denote the "kth perturbation" of f or $\boldsymbol{b}$.

Lemma 5.
Let $n \geq 1$ and $k = 1, \ldots, n$:

(i) $\|f(\boldsymbol{b}_{(k)}) - f(\boldsymbol{b}_{(k-1)})(\widetilde{\boldsymbol{b}}_k)\| \leq \delta_k \boldsymbol{b} \cdot \|\partial_k f(\boldsymbol{b}_{(k-1)})\| \cdot \widetilde{\boldsymbol{\beta}}_k$.

(ii) $\|f(\boldsymbol{b}_{(k-1)})(\widetilde{\boldsymbol{b}}_k) - \widetilde{f}(\widetilde{\boldsymbol{b}}_{(k)})\| \leq \delta_{k-1} f \cdot \widetilde{\boldsymbol{\beta}}_k$.

(iii) $\delta_k f \leq \left[\delta_k \boldsymbol{b} \cdot \|\partial_k f(\boldsymbol{b}_{(k-1)})\| + \delta_{k-1} f\right] \cdot \widetilde{\boldsymbol{\beta}}_k$.

We now have a recursive bound $\|\delta_n f\|$. But we need to convert the bound to only depend on the data $\|\boldsymbol{b}\|, \|f\|, \delta_k \boldsymbol{b}$. In particular, we remove any occurrences of $\partial_k f_{\boldsymbol{\alpha}}$ with the help of the next lemma:

Lemma 6. *For $k = 1, \ldots, n$:*
(i) $\|f(\boldsymbol{b}_{(k)})\| \leq \|f\| \cdot \prod_{i=1}^{k} \boldsymbol{\beta}_i$
(ii) For $\boldsymbol{\alpha} \in Supp_k(f)$,

$$\left\| \partial_k f_{\boldsymbol{\alpha}}(\boldsymbol{b}_{(k-1)}) \right\| \leq \boldsymbol{d}_k \cdot \|f_{\boldsymbol{\alpha}}(\boldsymbol{b}_{(k-1)})\|.$$

(iii) $\|\partial_k f(\boldsymbol{b}_{(k-1)})\| \leq \boldsymbol{d}_k \cdot \|f\| \cdot \prod_{i=1}^{k-1} \boldsymbol{\beta}_i$

Putting it all together:

Theorem 2. *For $k = 1, \ldots, n$,*

$$\delta_k f \leq \Big[\delta_0 f + \|f\| \cdot \sum_{i=1}^{k} \boldsymbol{d}_i \cdot \delta_i \boldsymbol{b}\Big] \cdot \Big(\prod_{i=1}^{k} \widetilde{\boldsymbol{\beta}}_i\Big).$$

The next lemma answers the question: given $\delta_L > 0$, how can we ensure that

$$\delta_{n-1} f := \|f(\boldsymbol{b}_{(n-1)}) - \widetilde{f}(\widetilde{\boldsymbol{b}}_{(n-1)})\|$$

is upper bounded by δ_L?

Lemma 7.
Given $\delta_L > 0$, $f, \widetilde{f} \in \mathbb{C}[\boldsymbol{z}]$ and $\boldsymbol{b}, \widetilde{\boldsymbol{b}} \in \mathbb{C}^{n-1}$ where $n > 1$.
Let $d = \max(deg_{\boldsymbol{z}_i}(f))$ and $M = \|\boldsymbol{b}\| + 1$.
If

$$\delta_f \leq \frac{\delta_L}{2((d+1)M^d)^{n-1}} \qquad (*)$$

and

$$\delta_{\boldsymbol{b}} \leq \min(1, \frac{\delta_L}{2d\|f\|(n-1)((d+1)M^d)^{n-1}}), \qquad (**)$$

then

$$\delta_{n-1} f \leq \delta_L.$$

4 Clustering for Triangular Systems

We now present our algorithm for solving the LCP for a given triple $(\mathbf{f}, \mathbf{B}^0, \epsilon)$, where $\epsilon > 0$. Instead of ϵ, we use $L := \lceil \log_2(1/\epsilon) \rceil$. $\mathbf{B}^0 = \mathbf{B}^0_{(n)}$ is the ROI (a polybox). $\mathbf{f} = \mathbf{f}_{(n)}$ is a triangular map with 0-dimensional set of zeros. We give our algorithm in the case where each $\mathbf{f}_i$ is known exactly; it can be generalized for oracle polynomials. It is based on the one-dimensional clustering algorithm (see Prop. Proposition 2), that proceeds by subdividing the initial ROI. The key stone in such a subdivision algorithm is a test that counts the number of roots with multiplicity in a disk. In the one-dimensional case, it is done with the so-called Pellet's test. Here we use this test with interval polynomials to compute towers. The main objects manipulated in our algorithms are cluster oracles, and their generalization when $n > 1$.

Cluster oracles in dimension $n \geq 1$. A polybox $\mathbf{B} \subseteq \mathbb{C}^n$ is called an ℓ-*tower* if there exists an ℓ-vector $\mathbf{m}_{(\ell)}$ such that $(\mathbf{B}_{(\ell)}, \mathbf{m}_{(\ell)})$ is a tower. A cluster oracle $\mathcal{O}$ in dimension $n > 1$ is defined to be a triple

$$\mathcal{O} = \langle \ell, \mathbf{B}, \mathbf{L} \rangle = \langle level(\mathcal{O}), domain(\mathcal{O}), precision(\mathcal{O}) \rangle$$

where $\ell \in \{0, \ldots, n\}$ is called the *level*, $\mathbf{B}$ is a polybox called the *domain* and $\mathbf{L}$ is a vector of integers called the *precision*. We will guarantee that if $level(\mathcal{O}) \geq 1$, $domain(\mathcal{O})$ is an ℓ-tower and $r(\Delta(domain(\mathcal{O})_i)) \leq 2^{-\mathbf{L}_i}$. The multiplicity information is implicitly carried out by a cluster oracle.

Cluster oracles at level 1. We generalize the *ClusterOracle* algorithm in Proposition 2 to $ClusterOracle1(\mathbf{f}, \mathcal{O})$, which returns a set $\{\mathcal{O}^1, \ldots, \mathcal{O}^k\}$ of cluster oracles at level 1. If $L = precision(\mathcal{O})_1$ and $\mathbf{B}^i$ is the domain of $\mathcal{O}^i$, then $(\Delta(\mathbf{B}^i))_1$ has radius at most 2^{-L}, and $\mathbf{B}^i_j = (domain(\mathcal{O}))_j$ for $j = 2, \ldots, n$. Moreover, the domains of these $\mathcal{O}^i$'s form a cover for the solution set of $\mathbf{f}$ in the domain of $\mathcal{O}$. All our oracles are subsequently descended from these $\mathcal{O}^i$'s.

Pellet test. Our goal is to "lift" a cluster oracle $\mathcal{O} = \langle \ell, \mathbf{B}, \mathbf{L} \rangle$ to one or more at level $\ell + 1$ (provided $\ell < n$) arising from subdividing $\mathbf{B}$. The fundamental tool for this purpose is the "Pellet test" and its variants (Graeffe-accelerated, soft-version, etc. – see [4,5,17]). Without distinguishing among these variants, we may describe a *generic Pellet test* denoted

$$T_*(\mathbf{f}_{\ell+1}, \mathbf{B}_{(\ell)}, \mathbf{B}_{\ell+1})$$

which returns an integer $m \geq -2$. $m \geq 0$ holds only if $m = \#(\Delta(\mathbf{B}_{\ell+1}), \mathbf{f}_{\ell+1}(\mathbf{B}_{(\ell)}))$, where $\mathbf{f}_{\ell+1}(\mathbf{B}_{(\ell)})$ is the univariate interval polynomial obtained by evaluating $\mathbf{f}_{\ell+1}$ on $\mathbf{B}_{(\ell)}$. If $m \geq 1$, then this implies that $\mathbf{B}$ is an $(\ell+1)$-tower. If $m = 0$ $\mathbf{B}$ does not contain zeros of $\mathbf{f}$. If $m = -1$ or $m = -2$, we say that the T_* test *failed*. These two modes of failure are important to understand for efficiency. Informally[8] $m = -1$ means the disc $\Delta(\mathbf{B}_{\ell+1})$ is not well-isolated (there are zeros near its boundary). In this case, the response is to subdivide $\mathbf{B}_{l+1}$. On the other hand, $m = -2$ means we need more accuracy in the evaluation $\mathbf{f}_{\ell+1}(\mathbf{B}_{(\ell)})$, which requires subdividing components $\mathbf{B}_i$ of $\mathbf{B}$ with $i < \ell$.

Lift of a natural cluster. The lifting process is performed by a function

$$ClusterOracleN(\mathbf{f}, \mathcal{O})$$

that takes in input a triangular map and a cluster oracle and outputs a pair $(flag, S)$ where $flag \in \{\mathbf{success}, \mathbf{failure}\}$ and S is a set of cluster oracles. It is

[8] The two failure modes may be traced to our soft comparison of real numbers $x : y$ (see [27,17]). It is reduced to the interval comparison $(x)_L : (y)_L$ for increasing L. If we can conclude $x > y$ or $x < y$, it is a success, else it is a failure. There are two failure modes: if we can conclude $\frac{1}{2}x < y < 2x$, this is a (-1)-failure (it is a potential "zero problem"). Otherwise it is a (-2)-failure (we repeat the interval test with larger L).

Algorithm 1 $ClusterTri(\mathbf{f}, \mathbf{B}, L)$

Input: A triangular map $\mathbf{f} = \mathbf{f}_{(n)}$, a ROI $\mathbf{B}^0 = \mathbf{B}^0_{(n)}$ and a precision $L > 1$.
Output: A set of cluster oracles at level n, that solves the LCP for $(\mathbf{f}, \mathbf{B}^0, 2^{-L})$.

```
1: Q.push(⟨0, B^0, (L, ..., L)⟩)          //initial cluster oracle at level 0
2: while Q contains cluster oracles at level less than n do
3:     O = ⟨ℓ, B, L⟩ ← Q.pop()            //assume ℓ < n
4:     if ℓ = 0 then
5:         Q.push(ClusterOracle1(f, O))
6:     else
7:         {flag, S} ← ClusterOracleN(f, O)
8:         if flag = success then
9:             Q.push(S)                  //S is a set of cluster oracles at level ℓ + 1
10:        else
11:            precision(O) ← (2L_(ℓ), L_{ℓ+1}, ..., L_n)
12:            level(O) ← 0
13:            Q.push(O)                  //O will be refined later
14: return Q
```

essentially the clustering algorithm depicted in [4]; it uses the T_*-test described above with $\ell = level(\mathcal{O})$ to count the number of roots in a disc. It returns the pair $(\textbf{failure}, \mathcal{O})$ when one T_*-test returns -2. When $ClusterOracleN(\mathbf{f}, \mathcal{O})$ returns $(\textbf{success}, S)$, then S is a list of pairwise disjoint cluster oracles at level $\ell + 1$ so that any solution in $\mathcal{O}$ is in a cluster oracle in S.

Solving the LCP problem. We are ready to present our main algorithm, called $ClusterTri(\mathbf{f}, \mathbf{B}, L)$, described in Algo. 1. It uses a queue Q to hold the active cluster oracles and lift these clusters level by level to level n. Let $\mathcal{O}$ be a cluster oracle in Q. If $level(\mathcal{O}) = 0$, $\mathcal{O}$ is lifted with $ClusterOracle1$ which returns a set S of cluster oracles at level 1 containing all the solutions in $\mathcal{O}$. If $level(\mathcal{O}) > 0$, $\mathcal{O}$ is lifted with $ClusterOracleN$, which may fail; in that case, the asked precision for levels less that ℓ of $\mathcal{O}$ is doubled and it's level is set to 0, this will force its refining in later executions of the **while** loop. When the lift of $\mathcal{O}$ succeeds, one obtains a set S of cluster oracles at level $\ell + 1$.

The correctness of $ClusterTri$ is a direct consequence of the correctness of $ClusterOracle1$ (see Prop. 2) and $ClusterOracleN$, and corollary 4.

The halting of $ClusterTri$ is a consequence of Lemma 7 (equation (**)) which shows that as long as the radius of $\Delta(\mathbf{B}_{(\ell)})$ approaches zero, Pellet test will eventually succeed; thus so does $ClusterOracleN$.

5 Implementation and benchmarks

We implemented in `Julia`[9] our complex solution clustering algorithm and made it available through the package `Ccluster.jl`[10]. It is named hereafter `tcluster`.

[9] https://julialang.org/
[10] https://github.com/rimbach/Ccluster.jl

It uses, as routine for clustering roots of univariate polynomials given by approximations, the univariate solver `ccluster` described in [17] and available in `Ccluster.jl`. The procedure for approximating a multivariate polynomial specialized in a cluster of fibers relies on the ball arithmetic library `arb` (see [18]), interfaced in `Julia` through the package `Nemo`[11].

Sec. 5.1 reports how `tcluster` performs on systems having clusters of solutions. Sec. 5.2 proposes benchmarks for solving random dense triangular systems with only regular solutions, and with solutions with multiplicities; `tcluster` is compared with three homotopy solvers. Sec. 5.3 is about using `tcluster` to cluster solutions of system triangularized with regular chains. Unless specified, `tcluster` is used with $\epsilon = 2^{-53}$. `tcluster` global (resp. local) holds for `tcluster` with initial box $\mathbf{B}$ centered in $\mathbf{0}$ with width 10^6 (resp. 2).

All the timings given below are sequential times in seconds on a Intel(R) Core(TM) i7-7600U CPU @ 2.80GHz machine with linux.

5.1 Clustering ability

Consider the triangular systems $\mathbf{g} = (f, g_2) = \mathbf{0}$ and $\mathbf{h} = (f, h_2) = \mathbf{0}$ where

$$\begin{array}{ll} f(z_1) & = z_1^{d_1} - (2^\delta z_1 - 1)^c \\ g_2(z_1, z_2) & = z_2^{d_2} z_1^{d_2} - 1 \\ h_2(z_1, z_2) & = z_2^{d_2} - z_1^{d_2} \end{array} \tag{5}$$

with $d_1 = 30$, $c = 10$, $\delta = 128$ and $d_2 = 10$. All the roots of f have multiplicity 1. A cluster S_1 of 10 roots is in a disk centered in $2^{-\delta}$ with radius $2^{-b} = 2^{-\frac{d_1\delta+\delta-1}{c}} \simeq 2^{-397}$ (see [21]). Since $d_1 > c > 1$, roots in S_1 have modulus $\leq 2^{-\delta} + 2^{-b} \leq 2^{-\delta+1} = 2^{-127} = \hat{\gamma}$. S_2 denotes the set of $d_1 - c$ others roots of f, that have a modulus of the order of $\gamma = 2^{\frac{c\delta}{d_1-c}} = 2^{64}$. The d_2 roots of g_2 are on a circle centered in 0 with radius $\geq \hat{\gamma}^{-1}$ when $z_1 \in S_1$, and of order γ^{-1} when $z_1 \in S_2$. The d_2 roots of h_2 are on a circle centered in 0 with radius $\leq \hat{\gamma}$ when $z_1 \in S_1$, and of order γ when $z_1 \in S_2$. All the solutions of $\mathbf{g} = \mathbf{0}$ and $\mathbf{h} = \mathbf{0}$ are included in the box $\mathbf{B}$ centered in $\mathbf{0}$ with width 10^{40}.

We computed clusters of solutions for the two systems with `tcluster` in $\mathbf{B}$ for four values of ϵ and reported the cluster structure as a sum where c_1 (respectively c_2, c_3) stands for the number of clusters with sum of multiplicities 1 (resp. 10, 100). Table. 1 gives this structure in columns #Sols, the solving time in columns t and the min and max precision required on clusters of f_1 in columns M and m (*i.e.* the log_2 of the radius of the disk isolating the clusters).

$(\mathbf{g} = \mathbf{0})$ has 20 clusters of 10 solutions above each root in S_2, where solutions have pairwise distance $\simeq 2^{-64}$. It has 10 clusters of 10 solutions above the cluster S_1 where solutions have pairwise distance $\leq 2^{-b} \simeq 2^{-397}$. This structure is found by `tcluster` with $\epsilon = 2^{-53}$. When $\epsilon = 2^{-106}$, the 20 clusters above roots in S_2 are split, not the ones above roots in S_1. When $\epsilon = 2^{-212}$, the clusters above roots in S_1 are split even if the pairwise distances between solutions in these

[11] http://nemocas.org/links.html

	$\mathbf{g} = \mathbf{0}$			$\mathbf{h} = \mathbf{0}$		
$log_2(\epsilon)$	#Sols	t (s)	(m,M)	#Sols	t (s)	(m,M)
-53	$0 + 30 \times 10$	0.17	(-212,- 424)	$200 + 0 \times 10 + 1 \times 100$	0.54	(-212, -212)
-106	$200 + 10 \times 10$	0.64	(-212,- 424)	$200 + 0 \times 10 + 1 \times 100$	0.57	(-212, -424)
-212	$300 + 0 \times 10$	3.91	(-424,- 848)	$200 + 10 \times 10 + 0 \times 100$	0.66	(-212, -848)
-424	$300 + 0 \times 10$	3.87	(-848,-1696)	$300 + 0 \times 10 + 0 \times 100$	3.78	(-848, -848)

Table 1. Clustering the solutions of systems defined in Sec. 5.1 with $d_1 = 30$, $c = 10$, $\delta = 128$, $d_2 = 10$ for four values of ϵ in box $\mathbf{B}$ centered in $\mathbf{0}$ with width 10^{40}.

clusters are far smaller than 2^{-212}; this is because isolating roots of g_2 with error less than $\epsilon = 2^{-212}$ requires more precision on roots of f, as shown is column (m,M). When $\epsilon = 2^{-424}$, all the clusters are split.

($\mathbf{h} = \mathbf{0}$) has 200 solutions above roots in S_2 and a cluster of 100 simple solutions above roots in S_1. The first (resp. second) components of the solutions in this cluster are in a disc of radius $\leq 2^{-b} \simeq 2^{-397}$ (resp. $\hat{\gamma} = 2^{-127}$). This cluster structure is found by `tcluster` with $\epsilon = 2^{-53}$ and $\epsilon = 2^{-106}$. When $\epsilon = 2^{-212}$, the cluster of 100 solutions is split in 10 clusters of 10 solutions. When $\epsilon = 2^{-424}$, the cluster is split in 100 solutions.

5.2 Benchmarks with random dense systems

We present benchmarks for randomly generated triangular systems without and with multiple solutions. We compare the efficiency and the robustness of `tcluster` and two homotopy solvers.

Homotopy solvers. Homotopy solving is a two-step process. First, an upper bound D (either the Bézout's bound, or a bound obtained with *polyhedral homotopy*, see [16]) on the number of solutions of the system is computed. Then D paths are followed to find the solutions. Among available homotopy solvers[12], we used in our benchmarks `HOM4PS-2.0`[13], `Bertini`[14] (see [2]) and `HomotopyContinuation.jl`[15] (hereafter, we denote it `HomCont.jl`). `HOM4PS-2.0` and `HomCont.jl` implement polyhedral homotopy, thus follow possibly less paths. `Bertini` and `HomCont.jl` compute the multiplicity structure of solutions. `Bertini` can use an Adaptive Multi-Precision (AMP) arithmetic; below `Bertini AMP` refers to `Bertini` with AMP.

Systems. We follow the approach of [8] to generate triangular systems with and without multiple solutions. The *type* of a triangular system $\mathbf{f}(\mathbf{z}) = \mathbf{0}$ with n equations is the list $(d_1, \ldots, d_n)$ where $d_i = deg_{z_i}(\mathbf{f}_i)$. A random dense polynomial $\mathbf{f}_i \in \mathbb{C}[z_1, \ldots, z_i]$ of degree d_i in z_i is generated as follows. If $i > 1$, $\mathbf{f}_i = \sum_{j=0}^{d_i} g_j z_i^j$

[12] other major homotopy solvers are `NAG4M2` (for `Macaulay2`), `PHCpack` and `HOM4PS-3`. `Bertini2` is still in development.

[13] http://www.math.nsysu.edu.tw/~leetsung/works/HOM4PS_soft.htm

[14] https://bertini.nd.edu/

[15] https://www.juliahomotopycontinuation.org/

where $g_j \in \mathbb{C}[z_1, \ldots, z_{i-1}]$ is a random dense polynomial of degree $d_i - j$ in z_{i-1}. $\mathbf{f}_1$ is a random dense polynomial in $\mathbb{C}[z_1]$ of degree d_1. A system $\mathbf{f}(\mathbf{z}) = \mathbf{0}$ of type $(d_1, \ldots, d_n)$ is obtained by generating successively random dense polynomials $\mathbf{f}_i$ of degrees d_i in z_i. Triangular systems with multiple solutions are obtained by taking $\mathbf{f}_1$ as above, and for $i = 2, \ldots, n$, $\mathbf{f}_i = a_i^2 (b_i z_i + c_i)^{\lfloor \frac{d_i+1}{2} \rfloor - \lfloor \frac{d_i}{2} \rfloor}$ where $a_i \in \mathbb{C}[z_1, \ldots, z_i]$ has degree $\lfloor \frac{d_i}{2} \rfloor$ in z_i and b_i, c_i are in $\mathbb{C}[z_1, \ldots, z_{i-1}]$ and have degrees d_i in z_{i-1}.

Benchmarks. In Table 2, we compare the three homotopy solvers and `tcluster` global and local on triangular systems with integer coefficients without and with multiple solutions. Coefficients of systems without multiple solutions are in $[-2^9, 2^9]$, while coefficients of systems with multiple solutions are in $[-2^{34}, 2^{34}]$. In both cases, we generated 5 systems of each type. Here `tcluster` global found all the solutions but in general this is not guaranteed. The columns #Sols give the average number of solutions counted with multiplicities found by each solver and the columns t the average time. The columns #Clus give the average number of clusters found by `tcluster`. The systems we generated have $d_1 \times \ldots \times d_n$ solutions which is the Bézout's bound, and the homotopy solvers have to follow this number of paths.

Systems with only simple solutions. For type (9,9,9,9,9), `Bertini AMP` has been stopped after 1 hour and `HOM4PS-2.0` terminates with a segmentation fault. Homotopy solvers should find all the solutions. `Bertini AMP` failed in this task for one system of type $(9, 9, 9, 9)$ and two systems of type $(2, 2, 2, 2, 2, 2, 2, 2, 2, 2)$ but acknowledged that solutions could be missing. `HOM4PS-2.0` returns incorrect results without warnings. In contrast, `tcluster` global always finds the correct number of solutions. `tcluster` global is in general faster than `Bertini AMP` and is faster than `HOM4PS-2.0` for systems of types $(6, 6, 6, 6, 6)$ and $(9, 9, 9, 9)$. For systems of highest degree polynomials, `tcluster` global and `HomCont.jl` present similar solving times. The timings for systems of type $(2, 2, 2, 2, 2, 2, 2, 2, 2, 2)$ emphasize that the efficiency of our solver is not penalized by high dimensional systems since it performs inductively subdivisions in boxes in $\mathbb{C}$. `tcluster` local is significantly faster than the other approaches.

Systems with multiple solutions. A well isolated multiple solution is reported by `tcluster` in a cluster with its multiplicity. In all cases, the number of clusters found by `tcluster` global is the number of distinct solutions of each systems. `HOM4PS-2.0` fails in finding all the solutions. `Bertini AMP` computes correctly the multiplicity of solutions. `HomCont.jl` fails in computing correctly the multiplicity structure of solutions. For type (9,9,9), `Bertini AMP` has been stopped after 1 hour. `tcluster` global is faster than `Bertini AMP` and `HomCont.jl`, and faster than `HOM4PS-2.0` for systems of type $(9, 9, 9)$ and $(6, 6, 6, 6)$.

5.3 Systems obtained by triangularization

In this subsection, we report on using `tcluster` for clustering the solutions of triangular systems $\mathbf{f}(\mathbf{z}) = \mathbf{0}$ obtained from a non-triangular system $\mathbf{g}(\mathbf{z}) = \mathbf{0}$

	tcluster local		tcluster global		HOM4PS-2.0		Bertini AMP		HomCont.jl	
type	#Sols, #Clus	t (s)	#Sols, #Clus	t (s)	#Sols	t (s)	#Sols	t (s)	#Sols	t (s)
Systems with only simple solutions										
(6,6,6)	34.2, 34.2	0.04	216, 216	0.35	216	0.06	216	1.17	216	2.77
(9,9,9)	149, 149	0.24	729, 729	1.43	713	0.47	729	29.3	729	4.21
(6,6,6,6)	63.4, 63.4	0.10	1296, 1296	2.21	1274	1.37	1296	24.2	1296	4.70
(9,9,9,9)	559, 559	1.06	6561, 6561	14.6	6036	111	6560	1605	6561	14.0
(6,6,6,6,6)	155, 155	0.37	7776, 7776	13.8	7730	28.6	7776	318	7776	11.5
(9,9,9,9,9)	1739, 1739	4.83	59049, 59049	130	-	-	?	>3600	59049	116
(2,2,2,2,2,2,2,2,2)	0, 0	0.13	1024, 1024	2.92	1024	2.74	1023	8.63	1024	4.84
Systems with multiple solutions										
(6,6)	10.8, 5.40	0.01	36, 18	0.06	36	0.00	18	3.63	17.4	1.74
(9,9)	23.8, 13.6	0.03	81, 45	0.17	67.4	0.06	45	218	33.6	3.27
(6,6,6)	35.2, 8.80	0.05	216, 54	0.26	210	0.16	54	47.9	53.2	2.75
(9,9,9)	113, 37.6	0.22	729, 225	1.10	357	18.9	?	>3600	159	28.4
(6,6,6,6)	81.6, 10.2	0.21	1296, 162	1.29	1010	4.46	162	662	134	8.06

Table 2. Solving random dense triangular systems with `tcluster`, `HOM4PS-2.0`, `Bertini AMP` and `HomCont.jl`.

with Regular Chains (RC, see [1,10]). Algorithms for triangularizing systems with RC produce a set of triangular systems $\{\mathbf{f}_1(\mathbf{z}) = \mathbf{0}, \ldots, \mathbf{f}_l(\mathbf{z}) = \mathbf{0}\}$ having distinct solutions whose union is the set of distinct solutions of $\mathbf{f}(\mathbf{z}) = \mathbf{0}$. The multiplicities of solutions are not preserved by this process.

Systems. We consider non-triangular systems $\mathbf{g}(\mathbf{z}) = \mathbf{0}$ both classical (coming from [7]), and sparse random where $\mathbf{g} = (g_1, \ldots, g_n)$ and each g_i has the form $g_i(\mathbf{z}) = z_i^{d_i} - g_i'(\mathbf{z})$ where g_i' is a polynomial in $\mathbb{Z}[\mathbf{z}]$ having total degree $d_i - 1$, integers coefficients in $[-2^8, 2^8]$ and 5 monomials. The *type* of such a system is the tuple $(d_1, \ldots, d_n)$. The set of all the examples can be found at https://cims.nyu.edu/~imbach/IPY19/IPY19.txt.

The benchmark. For several types, we generated a system as described above and computed a triangular systems with the `Maple` function `RegularChains[Triangularize]` with option `'probability'=0.9`. For the classical systems, we used no option. In table 3, column RC gives the time to compute the RCs. We solved the triangular systems of the obtained regular chains with `tcluster`; columns `tcluster` global report the number of solutions and solving time for `tcluster`. We also used the function `RootFinding[Isolate]` of `Maple` with options `digits=15, output=interval, method='RC'` (*i.e.* using regular chains) to solve our systems; columns Isolate RC report the number of real solutions and the solving time for Isolate. We also used `Bertini AMP` to solve the original systems; columns `Bertini AMP` report the number of paths followed (column #Paths), the solving time and the number of solutions with the multiplicity structure found by `Bertini`: $c_1 + c_2 \times m_2 + c_3 \times m_3$ means c_1 (respectively c_2, c_3) solutions with multiplicity 1 (resp. m_2, m_3). We also tested `HOM4PS-2.0` and `HomCont.jl` for these systems. The running time of `HOM4PS-2.0` is always less than 0.05s, but the number of solutions reported is wrong. `HomCont.jl` always finds the correct number of solutions but is slower than `Bertini AMP` except

	Bertini AMP			Isolate RC		RC	tcluster global	
type/name	#Sols	#Paths	t (s)	#Sols	t (s)	t (s)	#Sols	t (s)
Random systems								
(4,4,4)	64	64	0.06	6	7.53	3.82	64	0.80
(5,5,5)	125	125	0.30	?	>1000	24.2	125	6.89
(3,3,3,4)	108	108	0.13	?	>1000	52.4	108	3.42
(3,3,4,4)	144	144	0.26	?	>1000	68.7	144	8.59
Classical systems with only simple solutions								
Arnborg-Lazard	20	120	0.80	8	3.09	0.08	20	0.07
Czapor-Geddes-Wang	24	720	28.6	2	1.87	0.17	24	0.38
cyclic-5	70	120	0.35	10	1.92	0.55	70	0.71
Classical systems with multiple solutions								
5-body-homog	$45 + 2 \times 3 + 2 \times 24$	224	7.63	11	8.30	0.16	49	0.38
Caprasse	$24 + 8 \times 4$	144	0.25	18	1.49	0.24	32	0.12
neural-network	$90 + 18 \times 2$	162	0.36	22	5.82	0.13	108	0.56

Table 3. Solving non-triangular systems with regular chains and `tcluster`, and `Bertini AMP`.

for two systems for which polyhedral homotopy allows to reduce the number of paths to be followed. `HomCont.jl` solves *Czapor-Geddes-Wang* in 3.67 s and *5-body-homog* in 3.44 s.

Random systems in Table 3. Here the number of solutions is the Bézout's bound and `Bertini AMP` follows one path per solution. Homotopy solving in these cases is much more efficient than triangularizing the system with RC. The RC algorithm produces a triangular system of type $(d, 1, \ldots, 1)$ where d is the Bézout's bound with a huge bitsize: For the type $(3, 3, 4, 4)$, the triangular system has type $(144, 1, 1, 1)$ and each equation has bitsize about 738. `tcluster` has to isolate some solutions of the first equation at precision 2^{-424}. Solving the first equation with `ccluster` and $\epsilon = 2^{-424}$ takes 8.27s: `tcluster` spends most of the time in isolating roots of the first polynomial. Any improvement of `ccluster` will directly benefit to `tcluster`. For three of these systems, `RootFinding[Isolate]` has been stopped after 1000s.

Classical systems with only simple solutions in Table 3. These systems have few finite solutions compared to their Bézout's bounds, and `Bertini AMP` wastes time in following paths going to infinity. In contrast, `tcluster` is sensitive to the number of solutions in the initial solving domain. This explains why computing triangular systems and solving it with `tcluster` is faster than `Bertini AMP` for systems *Arnborg-Lazard*, *Czapor-Geddes-Wang*.

Classical systems with multiple solutions in Table 3. For these systems, `Bertini AMP` reports the multiplicity structure of the solutions. The triangularization step removes the multiplicity, and the RCs obtained are easier to solve; `tcluster` finds only clusters with one solution counted with multiplicity.

6 Future work

We presented an algorithm for computing clusters of complex solutions, together with multiplicity information, of triangular systems of polynomial equations. It is numerical and certified, it handles solutions with multiplicity and works locally. It can deal with systems whose equations are given by oracle polynomials. An implementation is publicly available and the experiments we carried out show the efficiency and robustness of our approach.

Our error analysis for the partial specialization of polynomials on algebraic numbers represented by oracle numbers is a first step towards a complexity analysis of our algorithm, which constitutes our future work. We would like to present such an analysis in terms of geometric parameters (*e.g.* separation of solutions) instead of only syntactic parameters (bit-size and degree).

References

1. Aubry, P., Lazard, D., Maza, M.M.: On the theories of triangular sets. Journal of Symbolic Computation **28**(1), 105 – 124 (1999)
2. Bates, D.J., Hauenstein, J.D., Sommese, A.J., Wampler, C.W.: Bertini: Software for numerical algebraic geometry. Available at bertini.nd.edu with permanent doi: dx.doi.org/10.7274/R0H41PB5
3. Batra, P.: Globally convergent, iterative path-following for algebraic equations. Math. in Computer Sci. **4**(4), 507–537 (2010)
4. Becker, R., Sagraloff, M., Sharma, V., Xu, J., Yap, C.: Complexity analysis of root clustering for a complex polynomial. In: Proceedings of the ACM on International Symposium on Symbolic and Algebraic Computation. pp. 71–78. ISSAC '16, ACM, New York, NY, USA (2016)
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. Beltrán, C., Leykin, A.: Certified numerical homotopy tracking. Experimental Mathematics **21**(1), 69–83 (2012)
7. Boulier, F., Chen, C., Lemaire, F., Moreno Maza, M.: Real root isolation of regular chains. In: Feng, R., Lee, W.s., Sato, Y. (eds.) Computer Mathematics. pp. 33–48. Springer Berlin Heidelberg, Berlin, Heidelberg (2014)
8. Cheng, J.S., Gao, X.S., Yap, C.K.: Complete numerical isolation of real roots in zero-dimensional triangular systems. Journal of Symbolic Computation **44**(7), 768–785 (2009)
9. Collins, G.E., Johnson, J.R., Krandick, W.: Interval arithmetic in cylindrical algebraic decomposition. Journal of Symbolic Computation **34**(2), 145 – 157 (2002)
10. Dahan, X., Maza, M.M., Schost, E., Wu, W., Xie, Y.: Lifting techniques for triangular decompositions. In: Proceedings of the 2005 International Symposium on Symbolic and Algebraic Computation. pp. 108–115. ISSAC '05, ACM, New York, NY, USA (2005)
11. Darboux, G.: Sur les développements en série des fonctions d'une seule variable. Journal de mathématiques pures et appliquées (Liouville Journal) **3**, II:291–312 (1876)

12. Dayton, B.H., Zeng, Z.: Computing the multiplicity structure in solving polynomial systems. In: Proceedings of the 2005 international symposium on Symbolic and algebraic computation. pp. 116–123. ACM (2005)
13. Dickenstein, A., Emiris, I. (eds.): Solving Polynomial Equations: Foundations, Algorithms, and Applications. Springer Berlin Heidelberg (2005)
14. Eigenwillig, A., Kettner, L., Krandick, W., Mehlhorn, K., Schmitt, S., Wolpert, N.: A Descartes Algorithm for Polynomials with Bit-Stream Coefficients. In: Ganzha, V., Mayr, E., Vorozhtsov, E. (eds.) CASC. LNCS, vol. 3718, pp. 138–149. Springer (2005)
15. Giusti, M., Lecerf, G., Salvy, B., Yakoubsohn, J.C.: On location and approximation of clusters of zeros: Case of embedding dimension one. Foundations of Computational Mathematics **7**(1), 1–58 (Feb 2007)
16. Huber, B., Sturmfels, B.: A polyhedral method for solving sparse polynomial systems. Mathematics of computation **64**(212), 1541–1555 (1995)
17. Imbach, R., Pan, V.Y., Yap, C.: Implementation of a near-optimal complex root clustering algorithm. In: Davenport, J.H., Kauers, M., Labahn, G., Urban, J. (eds.) Mathematical Software – ICMS 2018. pp. 235–244. Springer International Publishing, Cham (2018)
18. Johansson, F.: Arb: efficient arbitrary-precision midpoint-radius interval arithmetic. IEEE Transactions on Computers **66**, 1281–1292 (2017)
19. Kobel, A., Rouillier, F., Sagraloff, M.: Computing real roots of real polynomials ... and now for real! In: Proceedings of the ACM on International Symposium on Symbolic and Algebraic Computation. pp. 303–310. ISSAC '16, ACM, New York, NY, USA (2016)
20. Li, J., Cheng, J., Tsigaridas, E.: Local Generic Position for Root Isolation of Zero-dimensional Triangular Polynomial Systems. In: Koepf, W., E.Vorozhtsov (eds.) CASC 2012 - 14th International Workshop on Computer Algebra in Scientific Computing. Lecture Notes in Computer Science, vol. 7442, pp. 186–197. Springer, Maribor, Slovenia (Sep 2012)
21. Mignotte, M.: On the distance between the roots of a polynomial. Applicable Algebra in Engineering, Communication and Computing **6**(6), 327–332 (Nov 1995)
22. Moore, R.E., Kearfott, R.B., Cloud, M.J.: Introduction to interval analysis. Siam (2009)
23. Niang Diatta, D., Diatta, S., Rouillier, F., Roy, M.F., Sagraloff, M.: Bounds for polynomials on algebraic numbers and application to curve topology. arXiv e-prints arXiv:1807.10622 (Jul 2018)
24. Strzebonski, A., Tsigaridas, E.: Univariate real root isolation in an extension field and applications. Journal of Symbolic Computation **92**, 31 – 51 (2019)
25. Wampler, I.C.W., et al.: The Numerical solution of systems of polynomials arising in engineering and science. World Scientific (2005)
26. Xu, J., Burr, M., Yap, C.: An approach for certifying homotopy continuation paths: Univariate case. In: Proceedings of the 2018 ACM International Symposium on Symbolic and Algebraic Computation. pp. 399–406. ISSAC '18, ACM, New York, NY, USA (2018)
27. 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)
28. Zhang, Z., Fang, T., Xia, B.: Real solution isolation with multiplicity of zero-dimensional triangular systems. Science China Information Sciences **54**(1), 60–69 (2011)

Appendix : Error Analysis

This Appendix contains all the proofs for our error analysis. Section 3 is an excerpt.

Given $f, \widetilde{f} \in \mathbb{C}[\boldsymbol{z}]$ and $\boldsymbol{b}, \widetilde{\boldsymbol{b}} \in \mathbb{C}^n$, our basic goal is to bound the evaluation error

$$\|f(\boldsymbol{b}) - \widetilde{f}(\widetilde{\boldsymbol{b}})\|$$

in terms of $\delta_f := \|f - \widetilde{f}\|$ and $\delta_{\boldsymbol{b}} := \|\boldsymbol{b} - \widetilde{\boldsymbol{b}}\|$. This will be done by induction on n. Our analysis aims not just to produce some error bound, but to express this error in terms that are easily understood, and which reveals the underlying inductive structure. Towards this end, we introduce the following *β-bound* function: if d is a positive integer and $b \in \mathbb{C}$,

$$\beta(d, b) := \sum_{i=0}^{d} |b|^i. \tag{6}$$

A simple application of this β-bound is:

Lemma 8. *Let $b \in \mathbb{C}$ and $f \in \mathbb{C}[z]$. If d is the degree of f, then*

$$|f(b)| \leq \|f\| \cdot \beta(d, b), \qquad |f'(b)| \leq d\|f\| \cdot \beta(d-1, b).$$

Note that $\beta(d, b) \leq \max\left\{d+1, \frac{|b|^{d+1}-1}{|b|-1}\right\}$. We first treat the case $n = 1$. It will serve as the base for the inductive proof. Its proof requires a complex version of the Mean Value Theorem. Since this result is not well-known, we provide a statement and proof.

Theorem 3 (Complex Mean Value Theorem). *If $f : \mathbb{C} \to \mathbb{C}$ is holomorphic, then for any $a, b \in \mathbb{C}$,*

$$f(b) - f(a) = \omega \cdot (b - a) \cdot f'(\xi)$$

for some ξ in the line segment $[a, b]$ and some $\omega \in \mathbb{C}$ with $|\omega| \leq 1$.

Proof. This is a simple application of a similarly little known theorem of Darboux (1876) [11] which gives a finite Taylor expansion of f; see Bünger's formulation and proof in [3, Appendix]. For any $k \geq 1$, the theorem says

$$f(b) = \sum_{i=0}^{k-1} \frac{(b-a)^i}{i!} f^{(i)}(a) + \omega \frac{(b-a)^k}{k!} f^{(k)}(\xi)$$

for some ξ in the line segment $[a, b]$ and $\omega \in \mathbb{C}$ with $|\omega| \leq 1$. Choosing $k = 1$, $f(b) = f(a) + \omega(b-a)f'(\xi)$ or $f(b) - f(a) = \omega(b-a)f'(\xi)$. **Q.E.D.**

Corollary 9 (Complex Mean Value Inequality) *For all $a, b \in \mathbb{C}$, there is some $\xi \in [a, b]$ such that*

$$|f(b) - f(a)| \leq |b - a| \cdot |f'(\xi)|.$$

Lemma 10 (Case $n = 1$).
Let $f, \widetilde{f} \in \mathbb{C}[z]$, $b, \widetilde{b} \in \mathbb{C}$, and $\boldsymbol{d}(\widetilde{f}) \leq \boldsymbol{d}(f) \leq d$. If $\widetilde{f} = f \pm \delta_f$ and $\widetilde{b} = b \pm \delta_b$, then:

(i) $|f(b) - f(\widetilde{b})| \leq \delta_b \cdot \|f'\| \cdot \beta(d, |b| + \delta_b)$ *where f' is the differentiation of f.*
(ii) $|f(\widetilde{b}) - \widetilde{f}(\widetilde{b})| \leq \delta_f \cdot \beta(d, |b| + \delta_b)$
(iii) $|f(b) - \widetilde{f}(\widetilde{b})| \leq \left[\delta_f + \delta_b \cdot \|f'\|\right] \cdot \beta(d, |b| + \delta_b)$.

Proof.

(i) By the complex mean value inequality (Corollary 9):

$$\begin{aligned} |f(b) - f(\widetilde{b})| &\leq |b - \widetilde{b}| \cdot |f'(b \pm \delta_b)| \\ &\leq \delta_b \cdot \sum_{i=1}^{d} \left| i f_i (|b| + \delta_b)^{i-1} \right| \\ &\leq \delta_b \cdot \|f'\| \sum_{i=0}^{d-1} \left| (|b| + \delta_b)^i \right|. \end{aligned}$$

(ii) Also

$$\begin{aligned} |f(\widetilde{b}) - \widetilde{f}(\widetilde{b})| &= \left| \sum_{i=0}^{d} (f_i - \widetilde{f}_i) \widetilde{b}^i \right| \\ &\leq \delta_f \cdot \sum_{i=0}^{d} \left| \widetilde{b}^i \right| \\ &\leq \delta_f \cdot \beta(d, |b| + \delta_b). \end{aligned}$$

(iii) This follows from the triangular inequality

$$|f(b) - \widetilde{f}(\widetilde{b})| \leq |f(b) - f(\widetilde{b})| + |f(\widetilde{b}) - \widetilde{f}(\widetilde{b})|.$$

and the bounds in parts (i) and (ii).

Q.E.D.

The appearance of $\|f'\|$ in the above bound may be replaced by $d\|f\|$. Below, we develop similar bounds on partial derivatives in the multivariate case. For a general $n > 1$, we need to generalize the notations:

$$f, \widetilde{f} \in \mathbb{C}[\boldsymbol{z}], \qquad \boldsymbol{b}, \widetilde{\boldsymbol{b}} \in \mathbb{C}^n \tag{7}$$

satisfying $\widetilde{\boldsymbol{b}} = \boldsymbol{b} \pm \boldsymbol{\delta b}$ (i.e., $\widetilde{\boldsymbol{b}}_i = \boldsymbol{b}_i \pm \boldsymbol{\delta b}_i$ for each i). Let $\boldsymbol{d} = \boldsymbol{d}(f)$ (i.e., $\boldsymbol{d}_i = deg_{\boldsymbol{z}_i}(f)$ for each i). The *support* of f is $Supp(f) \subseteq \mathbb{N}^n$ where $f = \sum_{\boldsymbol{\alpha} \in Supp(f)} c_{\boldsymbol{\alpha}} \boldsymbol{z}^{\boldsymbol{\alpha}}$ where $c_{\boldsymbol{\alpha}} \in \mathbb{C} \setminus \{0\}$. Here, $\boldsymbol{z}^{\boldsymbol{\alpha}} := \prod_{i=1}^{n} \boldsymbol{z}_i^{\boldsymbol{\alpha}_i}$. We assume that $Supp(\widetilde{f}) \subseteq Supp(f)$. Our induction variable is $k = 1, \ldots, n$. For $\boldsymbol{\alpha} \in \mathbb{N}^n$, let $\pi_k(\boldsymbol{\alpha}) := (0, \ldots, 0, \boldsymbol{\alpha}_{k+1}, \ldots, \boldsymbol{\alpha}_n)$. E.g., if $k = n$ then $\pi_k(\boldsymbol{\alpha}) = \mathbf{0}$. Thus $\boldsymbol{\alpha} - \pi_k(\boldsymbol{\alpha}) = (\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_k, 0, \ldots, 0)$. Next define $Supp_k(f) := \{\pi_k(\boldsymbol{\alpha}) : \boldsymbol{\alpha} \in Supp(f)\}$. With this notation, we can write

$$f = \sum_{\boldsymbol{\alpha} \in Supp_k(f)} f_{\boldsymbol{\alpha}} \boldsymbol{z}^{\boldsymbol{\alpha}} \tag{8}$$

where each $f_{\boldsymbol{\alpha}} \in \mathbb{C}[\boldsymbol{z}_{(k)}]$. E.g., if $k = n$ then $Supp_k(f) = \{\mathbf{0}\}$ and so $f_{\mathbf{0}} = f$.

Running Example. Consider

$$f = xy + (x^3 - 1)y^2 z + (x^2 - y^2)z^3 \tag{9}$$

where $\boldsymbol{z} = (x, y, z)$. Then $Supp(f) = \{110, 321, 021, 203, 023\}$. We can represent f using the support $Supp_1(f) = \{010, 021, 003, 023\}$ as follows: $f = f_{010} \cdot y + f_{021} \cdot y^2 z + f_{003} \cdot z^3 f_{023} \cdot y^2 z^3$ where $f_{010} = x$, $f_{021} = x^3 - 1$, $f_{003} = x^2$, $f_{023} = -1$. Alternatively, using the support $Supp_2(f) = \{000, 001, 003\}$, we can write $f = f_{000} + f_{001} \cdot z + f_{003} \cdot z^3$ where $f_{000} = xy$, $f_{001} = (x^3 - 1)y^2$, $f_{003} = (x^2 - y^2)$.

Using (8), the partial specialization $f(\boldsymbol{b}_{(k)}) \in \mathbb{C}[z_{k+1}, \ldots, z_n]$ may be written

$$f(\boldsymbol{b}_{(k)}) = \sum_{\boldsymbol{\alpha} \in Supp_k(f)} f_{\boldsymbol{\alpha}}(\boldsymbol{b}_{(k)}) \cdot \boldsymbol{z}^{\boldsymbol{\alpha}}$$

It follows that

$$\|f(\boldsymbol{b}_{(k)})\| = \max_{\boldsymbol{\alpha} \in Supp_k(f)} \left| f_{\boldsymbol{\alpha}}(\boldsymbol{b}_{(k)}) \right|. \tag{10}$$

The k-th partial derivative is $\partial_k f := \frac{\partial f}{\partial z_k} = \sum_{\boldsymbol{\alpha} \in Supp_k(f)} (\partial_k f_{\boldsymbol{\alpha}}) \boldsymbol{z}^{\boldsymbol{\alpha}}$. Upon evaluation at $\boldsymbol{b}_{(k)}$, its norm is given by

$$\|\partial_k f(\boldsymbol{b}_{(k)})\| = \max_{\boldsymbol{\alpha} \in Supp_k(f)} \left| \partial_k f_{\boldsymbol{\alpha}}(\boldsymbol{b}_{(k)}) \right|. \tag{11}$$

Using our running example (8), let $k = 2$. Then $f = f_{000} + f_{001} \cdot z + f_{003} \cdot z^3$ with $f_{001} = xy$, $f_{001} = (x^3 - 1)y^2$, $f_{003} = (x^2 - y^2)$. Thus $\partial_2 f = x + (x^3 - 1)2y \cdot z - 2y \cdot z^3$. If $\boldsymbol{b}_{(2)} = (-1, 3)$, then $\|f(\boldsymbol{b}_{(2)})\| = \max\{3, 18, 8\} = 18$ and $\|\partial_2 f(\boldsymbol{b}_{(2)})\| = \max\{1, 12, 6\} = 12$.

We are ready for the generalization of Lemma 10. Assume that we are given $f, \widetilde{f} \in \mathbb{C}[\boldsymbol{z}] = \mathbb{C}[\boldsymbol{z}_{(n)}]$ and $\boldsymbol{b}, \widetilde{\boldsymbol{b}} \in \mathbb{C}$. Also the degree sequences satisfies $\boldsymbol{d}(\widetilde{f}) \leq \boldsymbol{d}(f)$, that is the inequality holds componentwise. Then we may define these quantities for $k = 1, \ldots, n$:

$$\begin{array}{lcll} \delta_k \boldsymbol{b} & := & |\boldsymbol{b}_k - \widetilde{\boldsymbol{b}}_k|, & \\ \delta_k f & := & \|f(\boldsymbol{b}_{(k)}) - \widetilde{f}(\widetilde{\boldsymbol{b}}_{(k)})\| & (\text{with } \delta_0 f = \|f - \widetilde{f}\|), \\ \boldsymbol{\beta}_k & := & \beta(\boldsymbol{d}_k, \boldsymbol{b}_k) & \\ \widetilde{\boldsymbol{\beta}}_k & := & \beta(\boldsymbol{d}_k, |\boldsymbol{b}_k| + \delta_k \boldsymbol{b}). & \end{array}$$

Note that δ_k is a operator that must attach to some function f or vector $\boldsymbol{b}$ to denote the "kth perturbation" of f or $\boldsymbol{b}$. We may restate Lemma 10(iii) using the new notations:

Corollary 11 *For a univariate f,*

$$\delta_1 f \leq \Big[\delta_0 f + \delta_1 \boldsymbol{b} \cdot \boldsymbol{d}_1 \cdot \|f\|\Big]\widetilde{\boldsymbol{\beta}}_1. \tag{12}$$

We now address the case of multivariate f:

Lemma 12 (=Lemma 5 in Text).
For $n \geq 1$ and each $k = 1, \ldots, n$:

$$\begin{aligned}
&\textit{(i)} \quad \|f(\boldsymbol{b}_{(k)}) - f(\boldsymbol{b}_{(k-1)})(\widetilde{\boldsymbol{b}}_k)\| \leq \delta_k \boldsymbol{b} \cdot \|\partial_k f(\boldsymbol{b}_{(k-1)})\| \cdot \widetilde{\boldsymbol{\beta}}_k. \\
&\textit{(ii)} \quad \|f(\boldsymbol{b}_{(k-1)})(\widetilde{\boldsymbol{b}}_k) - \widetilde{f}(\widetilde{\boldsymbol{b}}_{(k)})\| \leq \delta_{k-1} f \cdot \widetilde{\boldsymbol{\beta}}_k. \\
&\textit{(iii)} \quad \delta_k f \leq \Big[\delta_k \boldsymbol{b} \cdot \|\partial_k f(\boldsymbol{b}_{(k-1)})\| + \delta_{k-1} f\Big] \cdot \widetilde{\boldsymbol{\beta}}_k.
\end{aligned}$$

Proof. We note that (iii) amounts to adding the inequalities of (i) and (ii): specifically, $\delta_k f \leq \|f(\boldsymbol{b}_{(k)}) - f(\boldsymbol{b}_{(k-1)})(\widetilde{\boldsymbol{b}}_k)\| + \|f(\boldsymbol{b}_{(k-1)})(\widetilde{\boldsymbol{b}}_k) - \widetilde{f}(\widetilde{\boldsymbol{b}}_{(k)})\|$. Thus we only have to verify (i) and (ii). This will be shown by induction on k.

Suppose $k = 1$. This will be an application of Lemma 10(i) and (ii). We use the fact that $f = \sum_{\boldsymbol{\alpha} \in Supp_1(f)} f_{\boldsymbol{\alpha}} \boldsymbol{z}^{\boldsymbol{\alpha}}$, and $f(\boldsymbol{b}_{(k-1)}) = f(\boldsymbol{b}_{(0)}) = f$. Then (i) becomes

$$\begin{aligned}
\|f(\boldsymbol{b}_1) - f(\widetilde{\boldsymbol{b}}_1)\| &= \|\textstyle\sum_{\boldsymbol{\alpha} \in Supp_1(f)} (f_{\boldsymbol{\alpha}}(\boldsymbol{b}_1) - f_{\boldsymbol{\alpha}}(\widetilde{\boldsymbol{b}}_1)) \boldsymbol{z}^{\boldsymbol{\alpha}}\| \\
&= \max_{\boldsymbol{\alpha} \in Supp_1(f)} |f_{\boldsymbol{\alpha}}(\boldsymbol{b}_1) - f_{\boldsymbol{\alpha}}(\widetilde{\boldsymbol{b}}_1)| \\
&\leq \max_{\boldsymbol{\alpha} \in Supp_1(f)} \delta_1 \boldsymbol{b} \cdot \|f_{\boldsymbol{\alpha}}'\| \cdot \widetilde{\boldsymbol{\beta}}_1 \qquad \text{(by } \textit{Lemma } 10(i)\text{)} \\
&= \max_{\boldsymbol{\alpha} \in Supp_1(f)} \delta_1 \boldsymbol{b} \cdot \|\partial_1 f_{\boldsymbol{\alpha}}\| \cdot \widetilde{\boldsymbol{\beta}}_1 \\
&= \delta_1 \boldsymbol{b} \cdot \|\partial_1 f\| \cdot \widetilde{\boldsymbol{\beta}}_1.
\end{aligned}$$

Similarly, (ii) follows from

$$\begin{aligned}
\|f(\widetilde{\boldsymbol{b}}_1) - \widetilde{f}(\widetilde{\boldsymbol{b}}_1)\| &= \|\textstyle\sum_{\boldsymbol{\alpha} \in Supp_1(f)} (f_{\boldsymbol{\alpha}}(\widetilde{\boldsymbol{b}}_1) - \widetilde{f}_{\boldsymbol{\alpha}}(\widetilde{\boldsymbol{b}}_1)) \boldsymbol{z}^{\boldsymbol{\alpha}}\| \\
&= \max_{\boldsymbol{\alpha} \in Supp_1(f)} \Big|f_{\boldsymbol{\alpha}}(\widetilde{\boldsymbol{b}}_1) - \widetilde{f}_{\boldsymbol{\alpha}}(\widetilde{\boldsymbol{b}}_1)\Big| \\
&\leq \max_{\boldsymbol{\alpha} \in Supp_1(f)} \delta_{f_{\boldsymbol{\alpha}}} \cdot \widetilde{\boldsymbol{\beta}}_1 \qquad \text{(by } \textit{Lemma } 10(ii)\text{)} \\
&= \|f - \widetilde{f}\| \cdot \widetilde{\boldsymbol{\beta}}_1 \\
&= \delta_0 f \cdot \widetilde{\boldsymbol{\beta}}_1.
\end{aligned}$$

Suppose $k > 1$. We now prove (i). The left hand side (LHS) $\|f(\boldsymbol{b}_{(k)}) - f(\boldsymbol{b}_{(k-1)})(\widetilde{\boldsymbol{b}}_k)\|$ is the maximum of

$$|f_{\boldsymbol{\alpha}}(\boldsymbol{b}_{(k)}) - f_{\boldsymbol{\alpha}}(\boldsymbol{b}_{(k-1)})(\widetilde{\boldsymbol{b}}_k)| \tag{A}$$

where $\boldsymbol{\alpha}$ ranges over $Supp_k(f)$. We can rewrite (A) in the form $|f_{\boldsymbol{\alpha}}(\boldsymbol{b}_{(k-1)})(\boldsymbol{b}_k) - f_{\boldsymbol{\alpha}}(\boldsymbol{b}_{(k-1)})(\widetilde{\boldsymbol{b}}_k)|$. Applying Lemma 10(i), we can upper bound (A) by "$\delta_b \cdot \|f'\| \cdot \beta(d, |b|+\delta_b)$" where "$\delta_b$" here is $|\boldsymbol{b}_k - \widetilde{\boldsymbol{b}}_k| = \delta_k$, "$\|f'\|$" is $\|\partial_k f_{\boldsymbol{\alpha}}(\boldsymbol{b}_{(k-1)})\|$ and "$\beta(d, |b|+\delta_b)$" is $\boldsymbol{\beta}_k$. This establishes (i). Finally (ii) is proved by a similar invocation of Lemma 10(ii). **Q.E.D.**

We now have a recursive bound $\|\delta_n f\|$. But we need to convert the bound to only depend on the data $\|\boldsymbol{b}\|, \|f\|, \delta_k \boldsymbol{b}$. In particular, we remove any occurrences of $\partial_k f_{\boldsymbol{\alpha}}$ with the help of the next lemma:

Lemma 13 (= Lemma 6 in Text). *For $k = 1, \ldots, n$:*
(i) $\|f(\boldsymbol{b}_{(k)})\| \leq \|f\| \cdot \prod_{i=1}^{k} \boldsymbol{\beta}_i$
(ii) For $\boldsymbol{\alpha} \in Supp_k(f)$,

$$\left\| \partial_k f_{\boldsymbol{\alpha}}(\boldsymbol{b}_{(k-1)}) \right\| \leq \boldsymbol{d}_k \cdot \|f_{\boldsymbol{\alpha}}(\boldsymbol{b}_{(k-1)})\|.$$

(iii) $\|\partial_k f(\boldsymbol{b}_{(k-1)})\| \leq \boldsymbol{d}_k \cdot \|f\| \cdot \prod_{i=1}^{k-1} \boldsymbol{\beta}_i$

Proof. (i) The LHS of the inequality is equal to the maximum of $|f_{\boldsymbol{\alpha}}(\boldsymbol{b}_{(k)})|$ where $\boldsymbol{\alpha} \in Supp_k(f)$. First consider $k = 1$. In this case, $f_{\boldsymbol{\alpha}}$ is a univariate polynomial in $\boldsymbol{z}_1$ of degree at most $\boldsymbol{d}_1$, say $f_{\boldsymbol{\alpha}}(\boldsymbol{z}_1) = \sum_{i=0}^{\boldsymbol{d}_1} c_i \boldsymbol{z}_1^i$ where c is a coefficient of f. By Lemma 8,

$$|f_{\boldsymbol{\alpha}}(\boldsymbol{b}_1)| \leq \|f_{\boldsymbol{\alpha}}\| \boldsymbol{\beta}_1 \leq \|f\| \boldsymbol{\beta}_1,$$

proving the result for $k = 1$. For $k > 1$, each $f_{\boldsymbol{\alpha}}$ is a polynomial in $\boldsymbol{z}_{(k)}$, and we can write $f_{\boldsymbol{\alpha}}(\boldsymbol{b}_{(k)})$ as $f_{\boldsymbol{\alpha}}(\boldsymbol{b}_{(k-1)})(\boldsymbol{b}_k)$. By induction, the polynomial $f_{\boldsymbol{\alpha}}(\boldsymbol{b}_{(k-1)})(\boldsymbol{z}_k)$ has norm at most $\|f\| \cdot \prod_{i=1}^{k-1} \boldsymbol{\beta}_i$. Moreover, its degree is at most $\boldsymbol{d}_k$. So evaluating it at $\boldsymbol{b}_k$ gives a value of size at most $\|f\| \cdot \prod_{i=1}^{k} \boldsymbol{\beta}_i$.
(ii) Write $f_{\boldsymbol{\alpha}}(\boldsymbol{b}_{(k-1)}) = \sum_{i=0}^{\boldsymbol{d}_k} c_i \boldsymbol{z}_k^i$ where $c_i \in \mathbb{C}$ satisfies $|c_i| \leq \|f_{\boldsymbol{\alpha}}(\boldsymbol{b}_{(k-1)})\|$. Thus $\partial_k f_{\boldsymbol{\alpha}}(\boldsymbol{b}_{(k-1)})$ is a polynomial with norm

$$\left\| \partial_k f_{\boldsymbol{\alpha}}(\boldsymbol{b}_{(k-1)}) \right\| \leq \boldsymbol{d}_k \|f_{\boldsymbol{\alpha}}(\boldsymbol{b}_{(k-1)})\|.$$

(iii) Letting $\boldsymbol{\alpha}$ range over $Supp_k(f)$,

$$\begin{aligned}
\|\partial_k f(\boldsymbol{b}_{(k-1)})\| &= \max_{\boldsymbol{\alpha}} \|\partial_k f_{\boldsymbol{\alpha}}(\boldsymbol{b}_{(k-1)})\| \\
&\leq \max_{\boldsymbol{\alpha}} \boldsymbol{d}_k \cdot \|f_{\boldsymbol{\alpha}}(\boldsymbol{b}_{(k-1)})\| && \text{(from part (ii) second formula)} \\
&\leq \boldsymbol{d}_k \cdot \max_{\boldsymbol{\alpha}} \|f_{\boldsymbol{\alpha}}(\boldsymbol{b}_{(k-1)})\| \\
&\leq \boldsymbol{d}_k \cdot \|f(\boldsymbol{b}_{(k-1)})\| \\
&\leq \boldsymbol{d}_k \cdot \|f\| \cdot \prod_{i=1}^{k-1} \boldsymbol{\beta}_i && \text{(from part (i))}
\end{aligned}$$

Q.E.D.

Putting it all together:

Theorem 4 (=Theorem 2 in Text). *For $k = 1, \ldots, n$,*

$$\delta_k f \leq \Big[\delta_0 f + \|f\| \cdot \sum_{i=1}^{k} \boldsymbol{d}_i \cdot \delta_i \boldsymbol{b}\Big] \cdot \Big(\prod_{i=1}^{k} \widetilde{\boldsymbol{\beta}}_i\Big).$$

Proof. When $k = 1$, our formula is

$$\delta_1 f \leq \Big[\delta_0 f + \|f\| \cdot \boldsymbol{d}_1 \cdot \delta_1 \boldsymbol{b}\Big] \widetilde{\boldsymbol{\beta}}_1.$$

follows from the case $k = 1$ of Lemma 12(iii). For $k > 1$, we use induction:

$$\begin{array}{rll}
\delta_k f \leq & \Big[\delta_k \boldsymbol{b} \cdot \|\partial_k f(\boldsymbol{b}_{(k-1)})\| + \delta_{k-1} f\Big] \cdot \widetilde{\boldsymbol{\beta}}_k & \text{(By Lemma 12(iii))} \\
\leq & \Big[\delta_k \boldsymbol{b} \cdot \|\partial_k f(\boldsymbol{b}_{(k-1)})\| + \Big\{\delta_0 f + \|f\| \cdot \sum_{i=1}^{k-1} \boldsymbol{d}_i \cdot \delta_i \boldsymbol{b}\Big\} \cdot \Big(\prod_{i=1}^{k-1} \widetilde{\boldsymbol{\beta}}_i\Big)\Big] \cdot \widetilde{\boldsymbol{\beta}}_k & \text{(By induction)} \\
\leq & \Big[\delta_k \boldsymbol{b} \cdot \boldsymbol{d}_k \cdot \|f\| \cdot \Big(\prod_{i=1}^{k-1} \boldsymbol{\beta}_i\Big) + \Big\{\delta_0 f + \|f\| \cdot \sum_{i=1}^{k-1} \boldsymbol{d}_i \cdot \delta_i \boldsymbol{b}\Big\} \cdot \Big(\prod_{i=1}^{k-1} \widetilde{\boldsymbol{\beta}}_i\Big)\Big] \cdot \widetilde{\boldsymbol{\beta}}_k & \text{(By Lemma 13(iii))} \\
\leq & \Big[\delta_k \boldsymbol{b} \cdot \boldsymbol{d}_k \cdot \|f\| + \Big\{\delta_0 f + \|f\| \cdot \sum_{i=1}^{k-1} \boldsymbol{d}_i \cdot \delta_i \boldsymbol{b}\Big\}\Big] \cdot \Big(\prod_{i=1}^{k} \widetilde{\boldsymbol{\beta}}_i\Big) & \text{(since } \boldsymbol{\beta}_i \leq \widetilde{\boldsymbol{\beta}}_i\text{)} \\
= & \Big[\delta_0 f + \|f\| \cdot \sum_{i=1}^{k} \boldsymbol{d}_i \cdot \delta_i \boldsymbol{b}\Big] \cdot \Big(\prod_{i=1}^{k} \widetilde{\boldsymbol{\beta}}_i\Big). &
\end{array}$$

Q.E.D.

The next lemma answers the question: given $\delta_L > 0$, how can we ensure that

$$\delta_{n-1} f := \|f(\boldsymbol{b}_{(n-1)}) - \widetilde{f}(\widetilde{\boldsymbol{b}}_{(n-1)})\|$$

is upper bounded by δ_L?

Lemma 14 (=Lemma 7 in Text).
Given $\delta_L > 0$, $f, \widetilde{f} \in \mathbb{C}[\boldsymbol{z}]$ and $\boldsymbol{b}, \widetilde{\boldsymbol{b}} \in \mathbb{C}^{n-1}$ where $n > 1$.
Let $d = \max(deg_{\boldsymbol{z}_i}(f))$ and $M = \|\boldsymbol{b}\| + 1$.
If

$$\delta_f \leq \frac{\delta_L}{2((d+1)M^d)^{n-1}} \qquad (*)$$

and

$$\delta_{\boldsymbol{b}} \leq \min(1, \frac{\delta_L}{2d\|f\|(n-1)((d+1)M^d)^{n-1}}), \qquad (**)$$

then

$$\delta_{n-1} f \leq \delta_L.$$

Proof. Note that $\delta_f := \|f - \widetilde{f}\|$ in (*) and $\delta_{\boldsymbol{b}} := \|\boldsymbol{b} - \widetilde{\boldsymbol{b}}\|$ in (**). Since $\delta_{\boldsymbol{b}} \leq 1$, we conclude that $\|\widetilde{\boldsymbol{b}}\| \leq M$. Using the bounds $\boldsymbol{\beta}_k \leq \widetilde{\boldsymbol{\beta}}_k \leq (d+1)M^d$, the bound of Theorem 4 for the case $k = n - 1$ becomes

$$\begin{aligned}
\delta_{n-1} f &\leq \Big(\prod_{i=1}^{n-1} \widetilde{\boldsymbol{\beta}}_i\Big)\Big[\delta_f + \|f\| \cdot \sum_{i=1}^{n-1} \Big(\boldsymbol{d}_i \cdot \delta_i \boldsymbol{b}\Big)\Big] \\
&\leq \Big((d+1)M^d\Big)^{n-1}\Big[\delta_f + d\|f\|\delta_{\boldsymbol{b}}(n-1)\Big].
\end{aligned}$$

The inequalities (*) on δ_f, and (**) on $\delta_{\boldsymbol{b}}$, are designed to ensure that $\delta_{n-1} f \leq \delta_L$. **Q.E.D.**