Validated Root Enclosures for Interval Polynomials with Multiplicities

Florent Bréhard, Adrien Poteaux, Léo Soudant

▶ **To cite this version:**

Florent Bréhard, Adrien Poteaux, Léo Soudant. Validated Root Enclosures for Interval Polynomials with Multiplicities. ISSAC'23 : International Symposium on Symbolic and Algebraic Computation 2023, Jul 2023, Tromso, Norway. 10.1145/3597066.3597122 . hal-04138791

HAL Id: hal-04138791
https://hal.science/hal-04138791

Submitted on 23 Jun 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

Validated Root Enclosures for Interval Polynomials with Multiplicities

Florent Bréhard
Univ. Lille, CNRS, Centrale Lille, UMR 9189 CRIStAL
Lille, France
florent.brehard@univ-lille.fr

Adrien Poteaux
Univ. Lille, CNRS, Centrale Lille, UMR 9189 CRIStAL
Lille, France
adrien.poteaux@univ-lille.fr

Léo Soudant
École normale supérieure Paris-Saclay
Gif-sur-Yvette, France
soudant.leo@gmail.com

ABSTRACT

Twenty years ago, Zeng [28, 30] proposed floating-point algorithms to compute multiple roots of univariate polynomials with real or complex coefficients beyond the so-called "attainable accuracy barrier". Based on these foundations, we propose a validated numeric point of view on this problem. Our first contribution is an improvement of Zeng's multiplicity detection algorithm using a simple trick that allows us to recover much higher multiplicities. As our main contribution, we propose two floating-point validated algorithms to compute rigorous enclosures for multiple roots. They consist in carefully combining the ideas underlying Zeng's numerical algorithms with Newton-like fixed-point validation techniques. We also provide a prototype Julia implementation of these algorithms.

CCS CONCEPTS

• **Mathematics of computing** → **Interval arithmetic**; **Nonlinear equations**; **Solvers**.

KEYWORDS

multiple roots of univariate polynomials, fixed-point validation

1 MOTIVATION AND RELATED WORKS

This article is concerned with the numerical computation and validation of multiple roots of univariate polynomials.

Numerical multiple root-finding. Polynomial root-finding over real or complex numbers is one of the most common tasks in scientific computing and a major topic in numerical analysis and complexity theory, including nearly-optimal algorithms proposed in the 90s [14, 17]. The complexity of this problem [16] is however subject to the so-called "attainable accuracy barrier" in presence of multiple roots: generic root-finding algorithms executed on an input polynomial with coefficients truncated to $O(n)$ digits cannot retrieve more than $O(n/d)$ digits of accuracy for a root of multiplicity d. This could require multi-precision floating-point arithmetic, possibly far beyond the standard IEEE 53 bit double precision.

However, early remarks by Kahan [11] and Zeng's algorithmic contribution [28, 30] showed how to turn this *ill-conditioned* problem into a *well-conditioned* one, provided the polynomial's *multiplicity structure* $\boldsymbol{\ell} = (\ell_1, \dots, \ell_k) \in \mathbb{N}_{>0}^k$ is given or can be inferred.

Problem 1. *Given $p = p_n x^n + p_{n-1} x^{n-1} + \cdots + p_0 \in \mathbb{C}_n[x]$ and some maximum tolerated error on the input coefficients, recover the highest possible multiplicity structure $\boldsymbol{\ell} \in \mathbb{N}_{>0}^k$ and compute root approximations $\tilde{\boldsymbol{z}} \in \mathbb{C}^k$ such that $p \approx p_n \prod_{i=1}^k (x - \tilde{z}_i)^{\ell_i}$.*

Existing approaches for this problem include Schröder's modified Newton operator (see [7] and references therein) with quadratic convergence to a multiple root/a cluster of roots; or approximate GCDs (AGCD) of numerical polynomials [4], a notion closely related to multiple root-finding [1, 3, 5, 6, 23–25, 31]. Zeng's approach [28, 30] that inspires this article combines AGCD computations with Gauss-Newton iterations on overdetermined polynomial systems encoding the roots to overcome this attainable accuracy barrier.

Validated numerics. Starting with Moore's *interval arithmetic* [13] in the 60s, the field of *validated numerics* [15, 21, 27] aims at replacing heuristic or asymptotic error bounds with rigorous, explicitly computed enclosures. It relies on *set-valued* representations of mathematical objects, together with correctness-preserving algorithmic operations. For example, a real number x is represented by an interval $[x] = [\underline{x}, \overline{x}]$ with floating-point endpoints, i.e., $x \in [x]$. Arithmetic operations are defined on intervals to preserve this relation: if $x \in [x]$ and $y \in [y]$, then $[x] \star [y]$ for $\star \in \{+, -, \times, \div\}$ is built so that $x \star y \in [x] \star [y]$ holds. In particular, rounding errors are taken into account. Similar representations exist for complex numbers (which we will still refer to as intervals throughout this article), e.g., a pair of real intervals for the real and imaginary parts.

Most validated numerics libraries (e.g., INTLAB [19] or Arb [10]) can compute tight enclosures for simple roots of univariate polynomials. For multiple roots, a precise statement is needed to determine with respect to what the root enclosures are validated.

Problem 2. *Given $[p] = [p_n]x^n + \cdots + [p_0]$ with interval coefficients, $0 \notin [p_n]$ and $\boldsymbol{\ell} \in \mathbb{N}_{>0}^k$, compute intervals $[\boldsymbol{z}] = ([z_1], \dots, [z_k])$ such that for any polynomial $p \in [p]$ having multiplicity structure $\boldsymbol{\ell}$, there exist unique $z_i \in [z_i]$ such that $p = p_n \prod_{i=1}^k (x - z_i)^{\ell_i}$.*

The rigorous root enclosure methods of [20] are subject to the attainable accuracy barrier, since they are correct w.r.t. *all* polynomials $p \in [p]$, including those having clusters of roots.

Methods based on *deflation* [12, 22] consist in making an overdetermined system invertible by adding extra variables for perturbations on the coefficients of p. For instance, for a root z of multiplicity k, consider $\hat{p}(x) = p(x) + \varepsilon_{k-2} x^{k-2} + \cdots + \varepsilon_0$ and solve the system $\{\hat{p}^{(j)}(z) = 0, 0 \leqslant j < k\}$ for the variables $z, \varepsilon_0, \dots, \varepsilon_{k-2}$. The computed enclosure $[z]$ is guaranteed to contain a root of multiplicity k of an ε-perturbation of p. Such a property, however, is not per se a solution to Problem 2: the correcting perturbation $\hat{p} - p$ is not guaranteed to be the same for all distinct roots, the resulting $\hat{p}$ does not necessarily belong to $[p]$ and its coefficients could be close to those of $p \in [p]$ while having completely different roots.

Our contributions. They are twofold. First, after recalling the main lines of Zeng's two numerical algorithms GcdRoot and PejRoot in the beginning of Section 2, we improve the former (for multiplicity detection) using a simple trick. This gives Algorithm GcdRoot* in Section 2.3 which can correctly recover much higher multiplicities in our numerical experiments.

As our main contribution in Section 3, we propose an a posteriori validation method to simultaneously compute rigorous enclosures for the multiple roots, by carefully combining the ideas underlying Zeng's numerical algorithms with Newton-like fixed-point validation techniques. In particular, the combined nonlinear *and* overdetermined nature of Problem 2 is an obstacle for classical fixed-point validation. To overcome this, we design an original two-phase strategy based on two complementary validated algorithms ValGcdRoot and ValPejRoot. Our focus is on validating multiple roots beyond the attainable accuracy barrier: all our examples were executed in machine double precision to highlight this point.

Our implementation `ValidatedMultipleRoots.jl`[1] is a prototype open-source package in Julia [2]. It is based on Verzani's implementation of Zeng's algorithms in the `Polynomials.jl` package[2]. Although Zeng's original implementation in Matlab [29] is still available[3], we found more convenient to rely on this recent Julia code, notably for multi-precision and interval computations.

A note about complexity. We do not focus on the complexity of our algorithm: a complete complexity analysis would require a priori bounds for the precision needed to solve our problem, which would go beyond the scope and length of this paper. Nevertheless, if we stick to finite precision computations and only use classical matrix multiplication, it is easy to check that our algorithms run in complexity $\tilde{O}(k^2\, n)$, where n is the input degree and k the number of distinct roots. For instance, computations of the example shown in Table 1 are almost linear in n ($k = 4$ therein is constant).

2 ZENG'S ALGORITHM

Zeng's method to compute multiple roots of inexact polynomials [28, 30] mainly consists in numerically solving *overdetermined* systems of linear and nonlinear equations, in some least-squares sense. The necessary background is recalled in Section 2.1. Then we summarise the main ideas of the two algorithms GcdRoot and PejRoot composing Zeng's method in Section 2.2. After that, in Section 2.3, we present our first contribution, Algorithm GcdRoot*: a modification of GcdRoot that overcomes some of its limitations by reducing the impact of errors in the input.

2.1 Preliminaries.

The following paragraphs summarise notions such as pejorative manifolds, Sylvester matrices and approximate GCD. They make it possible to talk about multiple roots in a numerical context with precise mathematical definitions.

Pejorative manifolds. Given a multiplicity structure $\boldsymbol{\ell} \in \mathbb{N}_{>0}^k$ with $\ell_1 + \cdots + \ell_k = n$, the function $G_{\boldsymbol{\ell}}$ maps the roots $\boldsymbol{z} \in \mathbb{C}^k$ to the coefficients of the polynomial $p = \prod_{i=1}^k (x - z_i)^{\ell_i}$, i.e.,

$$G_{\boldsymbol{\ell}} \quad : \quad \boldsymbol{z} \in \mathbb{C}^k \quad \mapsto \quad \begin{pmatrix} g_0(\boldsymbol{z}) \\ \vdots \\ g_{n-1}(\boldsymbol{z}) \end{pmatrix} \in \mathbb{C}^n,$$
$$\text{s.t.} \quad p = x^n + g_{n-1}(\boldsymbol{z}) x^{n-1} + \cdots + g_0(\boldsymbol{z}).$$

Definition 2.1. The *pejorative manifold* $\Pi_{\boldsymbol{\ell}}$ is the image of the map $G_{\boldsymbol{\ell}} : \mathbb{C}^k \to \mathbb{C}^n$. It is a manifold (with singularities) of the coefficient space $\mathbb{C}^n$ parameterised by the roots $(z_1, \ldots, z_k) \in \mathbb{C}^k$.

The tangent space of $\Pi_{\boldsymbol{\ell}}$ at $\boldsymbol{z} \in \mathbb{C}^k$ is given by the Jacobian:

$$J_{G_{\boldsymbol{\ell}}}(\boldsymbol{z}) = \left(\boldsymbol{q_1} \;\middle|\; \cdots \;\middle|\; \boldsymbol{q_k} \right) \in \mathbb{C}^{n \times k}, \qquad \text{where } q_i := q_{i,n-1} x^{n-1} + \cdots + q_{i,0} := -\frac{\ell_i}{x - z_i} p. \tag{1}$$

The singularities of $\Pi_{\boldsymbol{\ell}}$ are the points where $J_{G_{\boldsymbol{\ell}}}(\boldsymbol{z})$ is rank deficient. This happens when at least two roots z_i and z_j coincide: the corresponding point belongs to the pejorative submanifold $\Pi_{\boldsymbol{\ell'}} \subseteq \Pi_{\boldsymbol{\ell}}$ associated to a higher multiplicity structure $\boldsymbol{\ell'} \in \mathbb{N}_{>0}^{k'}$ with $k' < k$.

As it will be seen in Section 2.2, Algorithm PejRoot of [28] relies on Gauss-Newton iterations to converge to a solution of $G_{\boldsymbol{\ell}}(\boldsymbol{z}) = \boldsymbol{p}$ on the pejorative manifold $\Pi_{\boldsymbol{\ell}}$.

Convolution and Sylvester matrices. Notions like divisibility and GCD in $\mathbb{C}[x]$ may seem ill-posed in a numerical context with errors or uncertain data. Yet, using convolution and Sylvester matrices, these questions are reduced to classical problems in linear algebra. They can be solved using efficient algorithms in numerical linear algebra, such as the singular value decomposition (SVD) [8, §2.5].

Let $p \in \mathbb{C}_n[x]$. For a given $k \geqslant 0$, the multiplication by p induces a morphism $\mathbb{C}_k[x] \to \mathbb{C}_{n+k}[x]$, the representation of which in the canonical bases is given by the k-th convolution matrix $C_k(p)$:

$$C_k(p) = \begin{pmatrix} p_0 & & \\ p_1 & \ddots & \\ \vdots & \ddots & p_0 \\ p_n & & p_1 \\ & \ddots & \vdots \\ & & p_n \end{pmatrix} \in \mathbb{C}^{(n+k+1)\times(k+1)}.$$

For two polynomials $p \in \mathbb{C}_n[x]$ and $q \in \mathbb{C}_m[x]$ of respective coefficients $\boldsymbol{p} \in \mathbb{C}^{n+1}$ and $\boldsymbol{q} \in \mathbb{C}^{m+1}$, the coefficients of pq are given by the vector $C_m(p)\, \boldsymbol{q} = C_n(q)\, \boldsymbol{p}$.

Definition 2.2. If $p \in \mathbb{C}_n[x]$, $q \in \mathbb{C}_m[x]$, and $0 \leqslant d \leqslant \min(n, m)$, then the d-th Sylvester matrix of (p, q) is

$$S_d(p, q) = \left(C_{m-d}(p) \;\middle|\; C_{n-d}(q) \right) \in \mathbb{C}^{(n+m-d+1)\times(n+m-2d+2)}.$$

$S_d(p, q)$ represents the map

$$(w, v) \in \mathbb{C}_{m-d}[x] \times \mathbb{C}_{n-d}[x] \mapsto wp + vq \in \mathbb{C}_{n+m-d}[x].$$

These matrices are connected with the GCD of p and q.

Lemma 2.3 ([31, Prop. 4]). *Let $p \in \mathbb{C}_n[x]$, $q \in \mathbb{C}_m[x]$, and u be their GCD. Denote by d the degree of u. Then $S_i(p, q)$ has full (column) rank for $d < i \leqslant \min(n, m)$, whereas $S_d(p, q)$ has a kernel of dimension 1 spanned by the vector $(\boldsymbol{w}, -\boldsymbol{v})$ corresponding to cofactors $w \in \mathbb{C}_{m-d}[x]$ and $v \in \mathbb{C}_{n-d}[x]$ such that $p = uv$ and $q = uw$.*

[1] https://gitlab.univ-lille.fr/florent.brehard/ValidatedMultipleRoots.jl/ – v. 0.0.1
[2] https://juliamath.github.io/Polynomials.jl/
[3] see "Supplemental Material" at https://dl.acm.org/doi/10.1145/992200.992209

The numerical treatment of Sylvester matrices underlies the key concept of approximate GCD.

Approximate GCD (AGCD). Computing the exact GCD of two polynomials p and q is an *ill-posed* problem, since under generic perturbations of the input, the result is always 1. The notion of *approximate GCD* [5] [4, §1] offers a robust definition by asking for the exact GCD of *perturbed* polynomials $p + \delta p$ and $q + \delta q$.

Definition 2.4. Let $p \in \mathbb{C}_n[x]$, $q \in \mathbb{C}_m[x]$ and $\varepsilon \geqslant 0$. We say that the polynomial u of degree $d \leqslant \min(n, m)$ is an ε-GCD for (p, q) if:

(i) u is the exact GCD of a pair $(p+\delta p, q+\delta q)$ with $\delta p \in \mathbb{C}_n[x]$, $\delta q \in \mathbb{C}_m[x]$, and $\|\delta \boldsymbol{p}\|_2^2 + \|\delta \boldsymbol{q}\|_2^2 \leqslant \varepsilon^2$;
(ii) No polynomial of degree larger than d satisfies condition (i);
(iii) u minimises $\|\delta \boldsymbol{p}\|_2^2 + \|\delta \boldsymbol{q}\|_2^2$ among degree-d polynomials satisfying (ii).

The method proposed in [5] relies on the observation that the GCD of $p + \delta p$ and $q + \delta q$ has degree at least d if and only if $S_d(p+\delta p, q+\delta q)$ is rank-deficient. Calling σ_{-1} the smallest singular value of $S_d(p, q)$ computed by SVD, they moreover prove that such an ε-perturbation $(\delta p, \delta q)$ exists only if $\sigma_{-1} \leqslant \varepsilon\sqrt{\max(n, m) - d + 1}$. Combining this necessary condition with a quadratic optimisation strategy to refine the AGCD u together with the cofactors v and w, the authors of [31] proposed a fully automated algorithm AGCD that computes the ε-GCD of p and q. This routine is a key ingredient of Algorithm GcdRoot of [28] summarised in Section 2.2 below.

2.2 Zeng's algorithm

Zeng's method is based on two algorithms called GcdRoot and PejRoot. GcdRoot infers the multiplicity structure of the input polynomial up to some user-defined tolerance, and computes root approximations with moderate accuracy. These roots serve as an initialiser for PejRoot which refines them to high accuracy.

2.2.1 Algorithm GcdRoot. It infers the multiplicity structure of $p \in \mathbb{C}_n[x]$ (together with root approximations) by inspecting the AGCD, defined in the previous section, of p and p'.

We now specialise Lemma 2.3 to the case $(p, q) = (p, p')$. For $1 \leqslant k \leqslant n$ the k-th discriminant matrix $\Delta_k(p)$ is:

$$\Delta_k(p) := S_{n-k}(p, p') = \left(C_{k-1}(p) \,\middle|\, C_k(p')\right) \in \mathbb{C}^{(n+k)\times(2k+1)}.$$

Lemma 2.5. *$p \in \mathbb{C}_n[x]$ has k distinct roots in $\mathbb{C}$ if and only if all $\Delta_i(p)$ for $1 \leqslant i < k$ have full column rank and $\Delta_k(p)$ has a kernel of dimension 1. This kernel is spanned by the vector $(\boldsymbol{w}, -\boldsymbol{v})$ corresponding to the coefficients of the cofactors*

$$v = \prod_{i=1}^{k}(x - z_i) \in \mathbb{C}_k[x], \quad w = \sum_{i=1}^{k} \ell_i \prod_{j\neq i}(x - z_j) \in \mathbb{C}_{k-1}[x], \quad (2)$$

where z_i are the roots of p with multiplicity ℓ_i. The polynomial

$$u = \prod_{i=1}^{k}(x - z_i)^{\ell_i - 1} \in \mathbb{C}_{n-k}[x],$$

is the GCD of p and p', and we have $uv = p$, $uw = p'$.

This result is used repeatedly (following somehow the symbolic square-free decomposition algorithm) to define algorithm GcdRoot of [30], that we briefly recall:

Algorithm 1 GcdRoot(p, ε)

Input: $p \in \mathbb{C}_n[x]$ and tolerance ε on the input error
Output: roots $\tilde{\boldsymbol{z}} \in \mathbb{C}^k$ together with multiplicities $\boldsymbol{\ell} \in \mathbb{N}_{>0}^k$

1: Compute the AGCD u of p and p' with cofactors v, w
2: Compute the (simple) roots $\tilde{\boldsymbol{z}}$ of v
3: Set $u^{(1)} = u$ and for $t = 2, \ldots$, compute $u^{(t)}$ as the AGCD of $u^{(t-1)}$ and ${u^{(t-1)}}'$ with cofactors $v^{(t)}, w^{(t)}$, until $\deg(u^{(t)}) = 0$
4: Compute the (simple) roots of all the $v^{(t)}$'s and match them to roots in $\tilde{\boldsymbol{z}}$.
5: Set $\ell_i := \min\{t \text{ s.t. } \tilde{z}_i \text{ is not a root of } v^{(t+1)}\}$
6: Return $\tilde{\boldsymbol{z}}, \boldsymbol{\ell}$

In [30], more details are given on how to propagate the error tolerances in the successive AGCD computations (step 3).

2.2.2 Algorithm PejRoot. The second algorithm proposed by Zeng in [28, 30], PejRoot, takes as input initial root approximations $\tilde{\boldsymbol{z}}_{\boldsymbol{0}}$ and the corresponding multiplicity structure $\boldsymbol{\ell}$. Such data is computed by GcdRoot, as previously seen.

Assume $p \in \mathbb{C}_n[x]$ is monic (otherwise rescale it by the inverse of its leading coefficient): $p = x^n + p_{n-1}x^{n-1} + \cdots + p_0$ with $\boldsymbol{p} \in \mathbb{C}^n$. $\boldsymbol{p}$ is supposed to lie on the pejorative manifold $\Pi_{\boldsymbol{\ell}}$, or nearby due to errors in the input coefficients of p. Moreover, the initial roots $\tilde{\boldsymbol{z}}_{\boldsymbol{0}}$ define an initial point $\boldsymbol{p_0} = G_{\boldsymbol{\ell}}(\tilde{\boldsymbol{z}}_{\boldsymbol{0}})$ on $\Pi_{\boldsymbol{\ell}}$.

Algorithm PejRoot consists in applying *Gauss-Newton iterations* to bring $\boldsymbol{p_0}$ closer and closer to $\boldsymbol{p}$ (or its projection onto $\Pi_{\boldsymbol{\ell}}$). The underlying system of equations,

$$G_{\boldsymbol{\ell}}(\boldsymbol{z}) = \boldsymbol{p} \quad \Leftrightarrow \quad \begin{cases} g_0(z_1, \ldots, z_k) = p_0, \\ \qquad\qquad \vdots \\ g_{n-1}(z_1, \ldots, z_k) = p_{n-1}, \end{cases} \quad (3)$$

consists of n polynomial equations in k variables z_i. Except in the case where all roots of p are simple, we have $k < n$ and (3) is *overdetermined*. Its Jacobian $J_{G_{\boldsymbol{\ell}}}(\boldsymbol{z})$, Eq. (1), is rectangular with full column rank. Instead of classical Newton iterations, which require a square, invertible Jacobian, PejRoot uses Gauss-Newton iterations:

$$\tilde{\boldsymbol{z}}_{\boldsymbol{t+1}} = \tilde{\boldsymbol{z}}_{\boldsymbol{t}} - J_{G_{\boldsymbol{\ell}}}(\tilde{\boldsymbol{z}}_{\boldsymbol{t}})^{+}\,(G_{\boldsymbol{\ell}}(\tilde{\boldsymbol{z}}_{\boldsymbol{t}}) - \boldsymbol{p}),$$

where $J_{G_{\boldsymbol{\ell}}}(\tilde{\boldsymbol{z}}_{\boldsymbol{t}})^{+}$ is the *Moore-Penrose inverse* (a.k.a. *pseudoinverse*) of $J_{G_{\boldsymbol{\ell}}}(\tilde{\boldsymbol{z}}_{\boldsymbol{t}})$. In practice, one merely computes the least-squares solution $\boldsymbol{h_t} \in \mathbb{C}^k$ of the overdetermined linear system:

$$J_{G_{\boldsymbol{\ell}}}(\tilde{\boldsymbol{z}}_{\boldsymbol{t}})\,\boldsymbol{h_t} = G_{\boldsymbol{\ell}}(\tilde{\boldsymbol{z}}_{\boldsymbol{t}}) - \boldsymbol{p}.$$

Proper convergence results may be found in [30, §3].

2.3 First contribution: improving GcdRoot

Larger errors in the coefficients of p of multiplicity structure $\boldsymbol{\ell} \in \mathbb{N}_{>0}^k$ increase the risk of the ε-GCD of p and p' having degree greater

Algorithm 2 PejRoot$(p, \boldsymbol{\ell}, \tilde{\boldsymbol{z}}_0)$

Input: $p \in \mathbb{C}_n[x]$, $\boldsymbol{\ell} \in \mathbb{N}_{>0}^k$ and initial root approx $\tilde{\boldsymbol{z}}_0 \in \mathbb{C}^k$
Output: refined root approximations $\tilde{\boldsymbol{z}} \in \mathbb{C}^k$

For $t = 0, 1, \ldots$:

1: Compute the least-squares solution $\boldsymbol{h_t} \in \mathbb{C}^k$ minimising the residue $d_t = \|J_{G_{\boldsymbol{\ell}}}(\tilde{\boldsymbol{z}}_t)\,\boldsymbol{h_t} - G_{\boldsymbol{\ell}}(\tilde{\boldsymbol{z}}_t) + \boldsymbol{p}\|_2$.
2: Update the approximation: $\tilde{\boldsymbol{z}}_{t+1} = \tilde{\boldsymbol{z}}_t - \boldsymbol{h_t}$.
3: If d_t is below some tolerance, return $\tilde{\boldsymbol{z}}_{t+1}$, otherwise continue.

than $n-k$. If so, GcdRoot infers a wrong multiplicity structure $\boldsymbol{\ell'}$, corresponding to a pejorative submanifold $\Pi_{\boldsymbol{\ell'}} \subset \Pi_{\boldsymbol{\ell}}$.

However, the repeated use of AGCD extraction in step 3 of GcdRoot is another significant source of errors that can be avoided if one can infer $\boldsymbol{\ell}$ directly, once the z_i have been computed. This is our first contribution: the procedure GcdRoot* based on the following simple observation to deduce the ℓ_i from the cofactors v, w.

Lemma 2.6. *If $\boldsymbol{z} = (z_1, \ldots, z_k)$ are the distinct multiple roots of $p \in \mathbb{C}_n[x]$, then the corresponding multiplicities ℓ_i are*

$$\ell_i = \frac{w(z_i)}{v'(z_i)},$$

where v and w are the cofactors as in Lemma 2.5.

The first-order sensitivity of this formula w.r.t. perturbations δv and δw on the coefficients of v and w is given by:

$$\delta\ell_i = \frac{v'(z_i)w'(z_i) - v''(z_i)w(z_i)}{v'(z_i)^3}(\delta v)(z_i) - \frac{w(z_i)}{v'(z_i)^2}(\delta v)'(z_i) + \frac{1}{v'(z_i)}(\delta w)(z_i) + O\left(\|\delta v\|^2 + \|\delta w\|^2\right).$$

Proof. The formula for ℓ_i directly follows from (2) in Lemma 2.5: we have $w(z_i) = \ell_i \prod_{j\neq i}(z_i - z_j)$ and $v'(z_i) = \prod_{j\neq i}(z_i - z_j)$.

Now suppose v and w are perturbed by δv and δw. The resulting perturbations δz_i of the roots z_i follow from $(v + \delta v)(z_i + \delta z_i) = 0$:

$$\delta z_i = \frac{(\delta v)(z_i)}{v'(z_i)} + O\left(\|\delta v\|^2\right).$$

By differentiating the relation for ℓ_i w.r.t. v, w, z_i and replacing δz_i, we obtain the desired estimate involving δv and δw. □

In particular, the sensitivity analysis shows that the technique used by Algorithm GcdRoot* to recover the multiplicity ℓ_i of the root z_i (line 3) is robust w.r.t. the numerical errors of v and w computed in line 1, except, as expected, if this root is too close to other roots of p (then $v'(z_i)$ tends to 0).

Numerical example. We compare our modified routine GcdRoot* to Zeng's original GcdRoot and show how this improves the total numerical multiple-root finding process solving Problem 1.

Example 1. Consider, as in [30, §4.6], the polynomial

$$p_m = (x-1)^{4m}(x-2)^{3m}(x-3)^{2m}(x-4)^m, \tag{4}$$

for $m = 1, 2, \ldots$, with coefficients truncated to standard double precision (53 bits). Table 1 summarises the multiplicity structures recovered by Zeng's GcdRoot and our GcdRoot*. GcdRoot fails as soon as $m = 5$, and the obtained ℓ_i rapidly diverge too much

Algorithm 3 GcdRoot*(p, ε)

Input: $p \in \mathbb{C}_n[x]$ and tolerance ε on the input error
Output: roots $\tilde{\boldsymbol{z}} \in \mathbb{C}^k$ together with multiplicities $\boldsymbol{\ell} \in \mathbb{N}_{>0}^k$

1: Compute the AGCD u of p and p' with cofactors v, w
2: Compute the (simple) roots $\tilde{\boldsymbol{z}}$ of v
3: Compute multiplicities ℓ_i as in Lemma 2.6, and round them to the nearest integer
4: Return $\tilde{\boldsymbol{z}}, \boldsymbol{\ell}$

m	GcdRoot	GcdRoot*	m	GcdRoot*
4	[16, 12, 8, 4]	[16, 12, 8, 4]	49	[196, 147, 98, **50**]
5	[**22**, 15, **9**, **4**]	[20, 15, 10, 5]	56	[224, 168, 112, **57**]
6	[**37**, **11**, **7**, **5**]	[24, 18, 12, 6]	59	[236, **178**, **119**, **58**]
7	[**47**, **8**, **10**, **5**]	[28, 21, 14, 7]	63	[252, 189, **127**, **62**]
8	[**56**, **8**, **11**, **5**]	[32, 24, 16, 8]	64	[256, **193**, **130**, **62**]
9	[**68**, **7**, **10**, **5**]	[36, 27, 18, 9]	65	[260, **195**, **129**, **66**]

Table 1: Multiplicity structures of p_m (4) recovered by GcdRoot and GcdRoot*. Boldface indicates incorrect results. The right part gives the first erroneous cases of GcdRoot*.

from the actual values. On the other hand, using the same tolerance parameters, GcdRoot* succeeds up to $m = 48$, and the first incorrectly recovered multiplicities (right part of the table) only differ from the actual ones by a few units.

Remark 2.7. In Zeng's original Matlab implementation [29], GcdRoot on the same example succeeds for m up to 7 and root approximations are more accurate. This is probably due to the fact that the use of rescaling weights in the AGCD procedure was not re-implemented in the more recent Julia package `Polynomials.jl` we rely on. It would be interesting to see the effect of weights on both GcdRoot and GcdRoot*.

About conditioning and numerical stability. The conditioning of Problem 1 for a given $\boldsymbol{\ell}$, i.e., the perturbation on the roots $\boldsymbol{z}$ induced by a multiplicity-preserving perturbation of the coefficients of p in system (3), is given by the norm of $J_{G_{\boldsymbol{\ell}}}(\boldsymbol{z})^+$, which is equal to the smallest singular value of $J_{G_{\boldsymbol{\ell}}}(\boldsymbol{z})$ [30, §3.4]. In a floating-point arithmetic context, we may prefer the *relative* structure-preserving condition number (see, e.g., [26, §12] or [9, §1.6]):

$$\kappa_{\boldsymbol{\ell}} := \frac{\|J_{G_{\boldsymbol{\ell}}}(\boldsymbol{z})^+\|}{\|\boldsymbol{z}\|/\|\boldsymbol{p}\|}.$$

Hence, knowing $\boldsymbol{\ell}$, the problem of computing multiple roots is not subject to infinite sensitivity: only *finitely* many bits are lost. Moreover, experiments in [30, §3.4] show that $\kappa_{\boldsymbol{\ell}}$ is moderate, even with high multiplicities, if the distinct roots are well separated.

However, rounding errors during execution also affect the computed roots. For PejRoot, each iteration solves a linear least-squares problem using the QR factorisation, which is backward stable [9, Thm. 20.3] but does not exploit the specific definition of $J_{G_{\boldsymbol{\ell}}}(\boldsymbol{z})$. Therefore, rounding errors grow with the matrix condition number $\kappa(J_{G_{\boldsymbol{\ell}}}(\boldsymbol{z})) = \|J_{G_{\boldsymbol{\ell}}}(\boldsymbol{z})\|\|J_{G_{\boldsymbol{\ell}}}(\boldsymbol{z})^+\|$. Similarly, the SVD used by the

AGCD routine in GcdRoot and GcdRoot* to compute the cofactors v and w is backward stable [26, Thm. 19.4]. Therefore, the rounding errors in v (from which the (simple) roots are extracted afterwards) grow with $\kappa(\Delta_k(p)) = \|\Delta_k(p)\|\|\Delta_k(p)^+\|$.

Although it is claimed in [30] that PejRoot is more accurate than GcdRoot, our experiments show it is not a general fact (see Table 3). A deeper numerical analysis of both methods would be necessary to understand in which cases one is better than the other.

3 VALIDATING MULTIPLE ROOTS

To solve Problem 2, we propose validation algorithms that compute rigorous error bounds on the numerical roots obtained previously. Algorithm ValPejRoot computes accurate bounds by translating Problem 2 into the straightforward, well-conditioned system (3) already used by PejRoot. Unfortunately, this system combines two difficulties: nonlinearity and overdetermination (see Section 3.3). The consequence is that ValPejRoot must rely on a priori root enclosures to compute tighter rigorous bounds. Such initial enclosures are computed by Algorithm ValGcdRoot, detailed in Section 3.2. It relies on a different system to encode the roots, inspired by Zeng's GcdRoot, whose structure makes it possible to depend on *no* a priori root enclosures. Before describing these two algorithms, which together solve Problem 2 (numerical examples are given in Section 3.4), we first provide preliminaries about interval arithmetic and a posteriori validation based on Newton-like fixed-point operators.

3.1 Preliminaries for the validation strategy.

Interval arithmetic. We call $\mathbb{IR}$ and $\mathbb{IC}$ the sets of real or complex intervals representable using floating-point arithmetic. Intervals are denoted using brackets: $[a]$, $[x]$, $[y]$, etc. Then $a \in [a]$ simply means that a, as an exact real or complex number, is contained in the mathematical set represented by $[a]$. These notations extend entrywise to vectors: $\boldsymbol{a} \in [\boldsymbol{a}]$ with $\boldsymbol{a} \in \mathbb{R}^n$ (or $\mathbb{C}^n$) and $[\boldsymbol{a}] \in \mathbb{IR}^n$ (or $\mathbb{IC}^n$) if $a_i \in [a_i]$ for all i. Analogous notations are used for polynomials (given by their coefficients) and matrices (given by their entries). Whenever a is already defined but not $[a]$, the latter denotes the singleton $\{a\}$ by default. Moreover, given $a \in \mathbb{R}$ or $\mathbb{C}$ and $r \in \mathbb{R}_{\geqslant 0}$, $[a \pm r]$ is the interval of radius r centred at a. $[\boldsymbol{a} \pm \boldsymbol{r}] \in \mathbb{IR}^n$ or $\mathbb{IC}^n$ is defined analogously for $\boldsymbol{a} \in \mathbb{R}^n$ or $\mathbb{C}^n$ and $\boldsymbol{r} \in \mathbb{R}^n_{\geqslant 0}$. The *magnitude* of $[a]$ is defined as $\operatorname{mag}[a] := \max\{|a|, a \in [a]\} \in \mathbb{R}_{\geqslant 0}$, and it is extended entrywise to vectors and matrices: $\operatorname{mag}[\boldsymbol{a}] \in \mathbb{R}^n_{\geqslant 0}$, $\operatorname{mag}[A] \in \mathbb{R}^{n\times k}_{\geqslant 0}$.

A function $[f] : \mathbb{IC}^k \to \mathbb{IC}$ is said to be an interval extension of $f : \mathbb{C}^k \to \mathbb{C}$ if for all $a_1 \in [a_1], \ldots, a_k \in [a_k]$, $f(a_1, \ldots, a_k) \in [f]([a_1], \ldots, [a_k])$. Most interval arithmetic libraries provide interval extensions for basic functions such as arithmetic operations $+, -, \times, \div$ and some elementary functions. They will be the building blocks of our validation algorithms.

Validation using Newton-like fixed-point operators. Let $H : \mathbb{C}^k \to \mathbb{C}^k$ be a system of equations, and let $\boldsymbol{x}^* \in \mathbb{C}^k$ denote an exact solution of $H(\boldsymbol{x}) = 0$. In many situations, $\boldsymbol{x}^*$ is not expressed as a finite composition of arithmetic operations and elementary functions, so that basic interval arithmetic is not sufficient to get an interval enclosure of it.

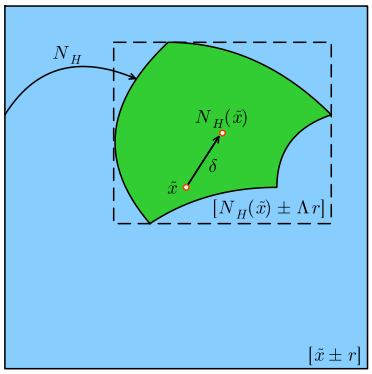

Figure 1: Theorem 3.2 in dimension $k = 2$, with the image (green solid shape) of the box $[\tilde{\boldsymbol{x}} \pm \boldsymbol{r}]$ contained in the dashed box $[N_H(\tilde{\boldsymbol{x}}) \pm \Lambda\,\boldsymbol{r}]$, thus satisfying inequality (6).

A posteriori validation consists in expressing $\boldsymbol{x}^*$ as a fixed point of a contracting operator, then deducing rigorous and tight error bounds for any input approximation $\tilde{\boldsymbol{x}}$ of $\boldsymbol{x}^*$. A popular option for locally invertible H is to use a Newton-like operator:

$$N_H(\boldsymbol{x}) := \boldsymbol{x} - A\,H(\boldsymbol{x}), \tag{5}$$

where $A \in \mathbb{C}^{k\times k}$ is an approximate inverse of the Jacobian $J_H(\tilde{\boldsymbol{x}})$ of H at $\tilde{\boldsymbol{x}}$. Then the zeros of H are exactly the fixed point of N_H (provided A is injective, which will be a byproduct of the validation). We will rigorously check that N_H is a *contraction* around $\tilde{\boldsymbol{x}}$.

Definition 3.1. N_H is said to be $\Lambda(\boldsymbol{r})$-Lipschitz over $[\tilde{\boldsymbol{x}} \pm \boldsymbol{r}] \in \mathbb{IC}^k$ with $\boldsymbol{r} \in \mathbb{R}^k_{\geqslant 0}$ if $\Lambda(\boldsymbol{r}) \in \mathbb{R}^{k\times k}_{\geqslant 0}$ satisfies:

$$|N_H(\boldsymbol{x}) - N_H(\boldsymbol{x'})| \leqslant \Lambda(\boldsymbol{r})\,|\boldsymbol{x} - \boldsymbol{x'}| \quad \text{for all } \boldsymbol{x}, \boldsymbol{x'} \in [\tilde{\boldsymbol{x}} \pm \boldsymbol{r}].$$

Such a Lipschitz matrix $\Lambda(\boldsymbol{r})$ for N_H can be rigorously computed by bounding the Jacobian of N_H over $\boldsymbol{x} \in [\tilde{\boldsymbol{x}} \pm \boldsymbol{r}]$:

$$J_{N_H}(\boldsymbol{x}) = I_k - A\,J_H(\boldsymbol{x}).$$

We also need to compute a rigorous bound $\boldsymbol{\delta} \in \mathbb{R}^k_{\geqslant 0}$ on the *defect*:

$$\boldsymbol{\delta} \geqslant |N_H(\tilde{\boldsymbol{x}}) - \tilde{\boldsymbol{x}}| = |A\,H(\tilde{\boldsymbol{x}})|.$$

Then the following generalisation of Banach's fixed-point principle (see e.g., [18, Thm. 1], or [21, Thm. 10.6] for an analogous statement due to Krawczyk) asserts the existence and uniqueness of a fixed point $\boldsymbol{x}^*$ in $[\tilde{\boldsymbol{x}} \pm \boldsymbol{r}]$.

Theorem 3.2. *If the following inequality holds entrywise:*

$$\Lambda(\boldsymbol{r})\,\boldsymbol{r} + \boldsymbol{\delta} < \boldsymbol{r}, \tag{6}$$

then N_H is a contraction *over $[\tilde{\boldsymbol{x}} \pm \boldsymbol{r}]$: the spectral radius (i.e., the modulus of the largest eigenvalue) $\lambda := \rho(\Lambda(\boldsymbol{r}))$ satisfies $\lambda < 1$, and N_H admits a unique fixed point $\boldsymbol{x}^*$ inside $[\tilde{\boldsymbol{x}} \pm \boldsymbol{r}]$.*

For the sake of completeness, we provide a routine ValidatedBounds which automates the process of finding a tight $\boldsymbol{r}$ satisfying (6) above. It is similar to Rohn's routine [18, §3] or Rump's ε-inflation method [21, Algo. 10.7], except that we here assume $\Lambda(\boldsymbol{r})$ to depend on $\boldsymbol{r}$, since we are concerned with nonlinear systems. Moreover Λ is assumed to be nondecreasing: if $\boldsymbol{r} \leqslant \boldsymbol{r'}$ (entrywise), then $\Lambda(\boldsymbol{r}) \leqslant \Lambda(\boldsymbol{r'})$ (entrywise).

Algorithm 4 ValidatedBounds$(\Lambda, \boldsymbol{\delta})$

Input: procedure $\boldsymbol{r} \in \mathbb{R}^k_{\geqslant 0} \mapsto \Lambda(\boldsymbol{r}) \in \mathbb{R}^{k\times k}_{\geqslant 0}$, and $\boldsymbol{\delta} \in \mathbb{R}^k_{\geqslant 0}$
Output: $\boldsymbol{r} \in \mathbb{R}^k_{\geqslant 0}$ satisfying $\Lambda(\boldsymbol{r})\boldsymbol{r} + \boldsymbol{\delta} < \boldsymbol{r}$, or failure message
Parameters: $\varepsilon = 10^{-10}, t_{\max} = 20$ (default values)

1: $\eta_i \leftarrow \begin{cases} \varepsilon\delta_i & \text{if } \delta_i > 0 \\ \varepsilon\delta_0 & \text{if } \delta_i = 0 \end{cases}$ for $i = 1 \dots k$, where $\delta_0 \leftarrow \min_{\delta_i > 0} \delta_i$
2: $\boldsymbol{\delta^+} \leftarrow \boldsymbol{\delta} + \boldsymbol{\eta}$
3: $\boldsymbol{r_0} \leftarrow 0 \in \mathbb{R}^k$
4: **for** $t = 0 \dots t_{\max} - 1$ **do**
5: $\quad \boldsymbol{r_{t+1}} \leftarrow \Lambda(\boldsymbol{r_t})\,\boldsymbol{r_t} + \boldsymbol{\delta^+}$
6: $\quad$ **if** $\boldsymbol{r_{t+1}} - \boldsymbol{r_t} < \boldsymbol{\eta}$ **then return** $\boldsymbol{r_t}$ **end if**
7: **end for**
8: **return** FAIL

Proposition 3.3. *If $\Lambda : \mathbb{R}^k_{\geqslant 0} \to \mathbb{R}^{k\times k}_{\geqslant 0}$ is continuous nondecreasing and $\boldsymbol{\delta} \neq 0 \in \mathbb{R}^k_{\geqslant 0}$, then* ValidatedBounds$(\Lambda, \boldsymbol{\delta})$ *satisfies:*

(i) If it succeeds in a finite number of iterations, then the computed vector $\boldsymbol{r}$ satisfies inequality (6).
(ii) If inequality (6) with $\boldsymbol{\delta}$ replaced by the ε-inflated vector $\boldsymbol{\delta^+}$ (see lines 1–2) admits a solution, then the algorithm computes a solution $\boldsymbol{r}$ satisfying (6) in a finite number of iterations.

Proof. The proof is similar to the linear case in [18, Thm. 2]. □

The overdetermined case. Particular care is needed when extending this strategy to the overdetermined case $H : \mathbb{C}^k \to \mathbb{C}^n$ with $k < n$. Linear overdetermined systems have been investigated in [18]. Algorithms ValGcdRoot and ValPejRoot in the following rely on overdetermined *nonlinear* systems (7) and (3).

First, the Jacobian $J_H(\tilde{\boldsymbol{x}}) \in \mathbb{C}^{n\times k}$ is still assumed to have full column rank, but it is no longer square and hence not invertible. Therefore, to define N_H (5), one takes for A a numerical approximation of its pseudoinverse:

$$A = J^+ = (J^*\,J)^{-1}\,J^* \in \mathbb{C}^{k\times n}, \quad \text{where } J = J_H(\tilde{\boldsymbol{x}}) \text{ and } J^* = \overline{J}^T.$$

Since $J^+J = I_k$, N_H is still contracting locally around $\tilde{\boldsymbol{x}}$, and ValidatedBounds provides rigorous bounds $\boldsymbol{r}$. However, A is no longer injective, implying that the fixed point $\boldsymbol{x^*}$ of N_H in $[\tilde{\boldsymbol{x}} \pm \boldsymbol{r}]$ (by Theorem 3.2) may no longer be a solution of $H(\boldsymbol{x}) = 0$. For instance, overdetermined systems have no solution in the generic case. This explains the careful phrasing of Propositions 3.5 and 3.6 below, and the need for a priori root enclosures for ValPejRoot.

3.2 Validated initial enclosures

Here we assume that $p \in \mathbb{C}_n[x]$ of multiplicity structure $\boldsymbol{\ell} \in \mathbb{N}^k_{>0}$ is represented by an interval polynomial $[p] \in \mathbb{IC}_n[x]$, and that approximations $\tilde{\boldsymbol{z}} \in \mathbb{C}^k$ of its roots have already been computed. Algorithm ValGcdRoot$([p], \boldsymbol{\ell}, \tilde{\boldsymbol{z}})$ constructs enclosures $[\boldsymbol{z}] \in \mathbb{IC}^k$ of its exact roots, based on an alternative system to Eq. (3). By Lemma 2.5, the roots of p are the simple roots of the square-free cofactor v:

$$wp - vp' = 0, \quad \text{and} \quad \prod_{i=1}^{k}(x - z_i) = v.$$

Let $\boldsymbol{v} \in \mathbb{C}^k$ and $\boldsymbol{w} \in \mathbb{C}^{k-1}$ denote the coefficients of v and w except the (fixed) leading ones: $v = x^k + v_{k-1}x^{k-1} + \cdots + v_0$, and $w = nx^{k-1} + w_{k-2}x^{k-2} + \cdots + w_0$. Form the vector $\boldsymbol{x} = (\boldsymbol{w}, -\boldsymbol{v}, \boldsymbol{z}) \in \mathbb{C}^{3k-1}$, and let $\boldsymbol{b} \in \mathbb{C}^{n+k-1}$ denote the coefficients of $x^k p' - nx^{k-1}p$. Then, with $\mathbf{1}$ denoting $(1, \dots, 1) \in \mathbb{N}^k_{>0}$, the system to solve is:

$$F_k(\boldsymbol{x}) = F_k\begin{pmatrix}\boldsymbol{w}\\-\boldsymbol{v}\\\boldsymbol{z}\end{pmatrix} = \begin{pmatrix}\boldsymbol{b}\\0\end{pmatrix} \Leftrightarrow \begin{cases}\Delta_{k-1}(p)\begin{pmatrix}\boldsymbol{w}\\-\boldsymbol{v}\end{pmatrix} = \boldsymbol{b},\\ G_{\mathbf{1}}(\boldsymbol{z}) - \boldsymbol{v} = 0.\end{cases} \tag{7}$$

This system has $n+2k-1$ equations in $3k-1$ variables. It is hence overdetermined too. However, the overdetermination entirely lies in the upper part of the system, which is linear. This is the key point making ValGcdRoot *not* relying on a priori root enclosures.

The Jacobian of F_k is the $(n + 2k - 1) \times (3k - 1)$ matrix:

$$J_{F_k}\begin{pmatrix}\boldsymbol{w}\\-\boldsymbol{v}\\\boldsymbol{z}\end{pmatrix} = \begin{pmatrix}\Delta_{k-1}(p) & O_{(n+k-1)\times k}\\ (O_{k\times(k-1)} \quad I_k) & J_{G_1}(\boldsymbol{z})\end{pmatrix}, \tag{8}$$

which is necessarily non-surjective when $k < n$. One builds the Newton-like validation operator:

$$N_{F_k}(\boldsymbol{x}) = \boldsymbol{x} - A\,\big(F_k(\boldsymbol{x}) - (\boldsymbol{b}, 0)\big), \tag{9}$$

where A is computed numerically as the pseudoinverse of $J_{F_k}(\boldsymbol{x})$ according to the following Lemma.

Lemma 3.4. *The Jacobian $J_{F_k}(\boldsymbol{x})$ is injective whenever $p \in \mathbb{C}_n[x]$ has k distinct roots z_i. Therefore, the pseudoinverse $J_{F_k}(\boldsymbol{x})^+$ satisfies $J_{F_k}(\boldsymbol{x})^+\,J_{F_k}(\boldsymbol{x}) = I_{3k-1}$, and it is equal to:*

$$J_{F_k}(\boldsymbol{x})^+ = \begin{pmatrix}A_1 & O_{(2k-1)\times k}\\ A_3 & A_2\end{pmatrix},$$
$$\text{where} \quad A_1 = \Delta_{k-1}(p)^+, \quad A_2 = J_{G_1}(\boldsymbol{z})^{-1},$$
$$A_3 = -A_2\,(O_{k\times(k-1)} \quad I_k)\,A_1 = -A_2\,(A_{1,ij})_{\substack{k\leqslant i\leqslant 2k-1\\ 1\leqslant j\leqslant n+k-1}}.$$

Proof. The injectivity of $J_{F_k}(\boldsymbol{x})$ follows from the one of $\Delta_{k-1}(p)$ by Lemma 2.5, together with the invertibility of $J_{G_1}(\boldsymbol{z})$. It automatically implies that $J_{F_k}(\boldsymbol{x})^+$ is a left inverse for $J_{F_k}(\boldsymbol{x})$. Finally, for any $\boldsymbol{c} \in \mathbb{C}^{n+2k-1}$, $\boldsymbol{y} = J_{F_k}(\boldsymbol{x})^+\,\boldsymbol{c}$ is by definition the least-squares solution of $J_{F_k}(\boldsymbol{x})\,\boldsymbol{y} = \boldsymbol{c}$. Since $J_{F_k}(\boldsymbol{x})$ is lower triangular by block and $J_{G_1}(\boldsymbol{z})$ is invertible, $\boldsymbol{y}$ is built by using $A_1 := S_{k-1}(p)^+$ to define its $(\boldsymbol{w}, -\boldsymbol{v})$ component (thus minimising the errors on the first $n + k - 1$ equations), and by adjusting the $\boldsymbol{z}$ component of $\boldsymbol{y}$ so as to cancel the remaining k equations. This leads to the given block matrix expression for $J_{F_k}(\boldsymbol{x})^+$. □

Once N_{F_k} is built, Algorithm ValGcdRoot proceeds by rigorously satisfying the hypotheses of Theorem 3.2 using interval arithmetic to deduce bounds $\boldsymbol{r} \in \mathbb{R}^{3k-1}_{\geqslant 0}$ for an approximation $\tilde{\boldsymbol{x}} = (\tilde{\boldsymbol{w}}, -\tilde{\boldsymbol{v}}, \boldsymbol{z})$ of system (7).

Proposition 3.5. *Let $[p] \in \mathbb{IC}_n[x]$, $\boldsymbol{\ell} \in \mathbb{N}^k_{>0}$ and $\tilde{\boldsymbol{z}} \in \mathbb{C}^k$. If* ValGcdRoot$([p], \boldsymbol{\ell}, \tilde{\boldsymbol{z}})$ *does not fail, then it computes enclosures $[\boldsymbol{z}] \in \mathbb{IC}^k$ such that for all $p \in [p]$ with multiplicity structure $\boldsymbol{\ell}$, there exists a unique vector of roots $\boldsymbol{z} \in [\boldsymbol{z}]$ such that $p = p_n \prod_{i=1}^k (x - z_i)^{\ell_i}$.*

Moreover, no $p \in [p]$ can have less than k distinct roots, and those having exactly k distinct ones have multiplicity structure $\boldsymbol{\ell}$.

Algorithm 5 ValGcdRoot($[p], \boldsymbol{\ell}, \tilde{z}$)

Input: interval polynomial $[p] \in \mathbb{IC}_n[x]$, multiplicity structure $\boldsymbol{\ell} \in \mathbb{N}^k_{>0}$, root approximations $\tilde{\boldsymbol{z}} \in \mathbb{C}^k$

Output: root enclosures $[z] \in \mathbb{IC}^k$ or failure message

1: compute $\tilde{v}, \tilde{w}$ as in (2) to form $\tilde{\boldsymbol{x}} = (\tilde{\boldsymbol{w}}, -\tilde{\boldsymbol{v}}, \tilde{\boldsymbol{z}})$
2: compute $A \approx J_{F_k}(\tilde{\boldsymbol{x}})^+$ numerically, as in Lemma 3.4
▷ *From now on, use interval arithmetic*
3: $\boldsymbol{\delta} \leftarrow \mathrm{mag}\left([A]\,([F_k](\tilde{\boldsymbol{x}}) - ([\boldsymbol{b}], 0))\right)$
4: $\Lambda : \boldsymbol{r} \in \mathbb{R}^{3k-1}_{\geqslant 0} \mapsto \mathrm{mag}\left(I_{3k-1} - [A]\,[J_{F_k}]([\tilde{\boldsymbol{x}} \pm \boldsymbol{r}])\right)$
5: $\boldsymbol{r} \leftarrow$ ValidatedBounds($\Lambda, \boldsymbol{\delta}$)
6: $[\boldsymbol{x}] = ([\boldsymbol{w}], -[\boldsymbol{v}], [\boldsymbol{z}]) \leftarrow [\tilde{\boldsymbol{x}} \pm \boldsymbol{r}]$
7: **for** $i = 1 \ldots k$ **do**
8: **if** $\frac{[w]([z_i])}{[v]'([z_i])} \cap \mathbb{N} \neq \{\ell_i\}$ **then return** FAIL **end if**
9: **end for**
10: **return** $[z]$

Proof. Given the candidate solution $\tilde{\boldsymbol{x}} = (\tilde{\boldsymbol{w}}, -\tilde{\boldsymbol{v}}, \tilde{\boldsymbol{z}})$ computed in line 1 and the Newton-like operator N_{F_k} (9) defined from the pseudoinverse A computed in line 2, the use of interval arithmetic beginning on line 3 guarantees that for any $p \in [p]$:
(1) $\boldsymbol{\delta}$ is a rigorous entrywise upper bound for the defect $N_{F_k}(\tilde{\boldsymbol{x}}) - \tilde{\boldsymbol{x}}$;
(2) $\Lambda(\boldsymbol{r})$ with arbitrary $\boldsymbol{r} \in \mathbb{R}_{\geqslant 0}$ is a rigorous entrywise bound for the Jacobian of N_{F_k} taken at any point inside the interval vector $[\tilde{\boldsymbol{x}} \pm \boldsymbol{r}]$, hence a valid Lipschitz matrix for N_{F_k} over this domain.

According to Theorem 3.2 and Proposition 3.3, the bounds $\boldsymbol{r}$ computed in line 5 (if ValidatedBounds does not fail), ensure that the spectral radius of $\Lambda(\boldsymbol{r})$ is smaller than 1, that the Newton-like operator N_{F_k} (for any $p \in [p]$) is a contraction over the $\boldsymbol{r}$-inflated interval vector $[\boldsymbol{x}] \in \mathbb{IC}^{3k-1}$, and therefore admits a unique fixed point $\boldsymbol{x}^* = (\boldsymbol{w}^*, -\boldsymbol{v}^*, \boldsymbol{z}^*)$ in it. Moreover, since $\Lambda(\boldsymbol{r})$ has the form $\begin{pmatrix} \Lambda_1 & O_{(2k-1)\times k} \\ \Lambda_3 & \Lambda_2(\boldsymbol{r}) \end{pmatrix}$ (with Λ_1, Λ_3 independent of $\boldsymbol{r}$), $\rho(\Lambda(\boldsymbol{r})) < 1$ implies that $\rho(\Lambda_1) < 1$ and $\rho(\Lambda_2(\boldsymbol{r})) < 1$, so that $A_1\,\Delta_{k-1}(p)$ and $A_2\,J_{G_1}(\boldsymbol{z})$ are invertible (hence also A_2 since it is a square matrix).

Let us now assume that $p \in [p]$ has the desired multiplicity structure $\boldsymbol{\ell}$. Then by Lemma 2.5, there are unique polynomials v and w of the form $v = x^k + v_{k-1}x^{k-1} + \cdots + v_0$ and $w = nx^{k-1} + w_{k-2}x^{k-2} + \cdots + w_0$ satisfying $wp - vp' = 0$. The corresponding coefficients therefore satisfy $\Delta_{k-1}(p) \begin{pmatrix} \boldsymbol{w} \\ -\boldsymbol{v} \end{pmatrix} = \boldsymbol{b}$, and since $A_1\,\Delta_{k-1}(p)$ is invertible, they are equal to $(\boldsymbol{w}^*, -\boldsymbol{v}^*)$. Now from the last k equations of $A\left(F_k(\boldsymbol{x}^*) - (\boldsymbol{b}, 0)\right) = 0$,

$$A_3 \underbrace{\left(\Delta_{k-1}(p) \begin{pmatrix} \boldsymbol{w}^* \\ \boldsymbol{v}^* \end{pmatrix} - \boldsymbol{b}\right)}_{=0} + A_2\left(G_1(\boldsymbol{z}^*) - \boldsymbol{v}^*\right) = 0,$$

we deduce that $G_1(\boldsymbol{z}^*) = \boldsymbol{v}^*$, by invertibility of A_2. Hence, $\boldsymbol{z}^*$ are the (simple) roots of v, which are also the (multiple) roots of p.

Finally, the tests in lines 7–9 carried out in interval arithmetic guarantee that the obtained roots are matched with the correct multiplicities, following Lemma 2.6.

Now for the second claim, if $p \in [p]$ has less than k distinct roots, then $\Delta_{k-1}(p)$ is rank deficient, and so is the Jacobian J_{F_k}. Therefore $\rho(\Lambda(\boldsymbol{r})) \geqslant 1$ and line 5 will necessarily raise an error. □

3.3 Refined validated enclosures

Given a polynomial $p \in \mathbb{C}_n[x]$ of multiplicity structure $\boldsymbol{\ell} \in \mathbb{N}^k_{>0}$, represented by $[p] \in \mathbb{IC}_n[x]$, Algorithm ValPejRoot aims at validating approximate roots $\tilde{\boldsymbol{z}} \in \mathbb{C}^k$, seen as an approximate solution of system (3). The Newton-like validation operator,

$$N_{G_{\boldsymbol{\ell}}}(\boldsymbol{z}) := \boldsymbol{z} - A\,\left(G_{\boldsymbol{\ell}}(\boldsymbol{z}) - \boldsymbol{p}\right), \tag{10}$$

is built from a numerical approximation A of the pseudoinverse $J_{G_{\boldsymbol{\ell}}}(\tilde{\boldsymbol{z}})^+ \in \mathbb{C}^{k\times n}$.

Using interval arithmetic and Theorem 3.2, ValPejRoot constructs enclosures $[z] \in \mathbb{IC}^k$ around $\tilde{\boldsymbol{z}}$ containing a unique fixed point $\boldsymbol{z}^*$ of $N_{G_{\boldsymbol{\ell}}}$. However, since A is not injective, one cannot deduce that $\boldsymbol{z}^*$ is a solution of system (3). Moreover, the argument used in the proof of Proposition 3.5 for ValGcdRoot was specific to the structure of system (7), whose overdetermined part was *linear*. This is why ValPejRoot($[p], \boldsymbol{\ell}, \tilde{\boldsymbol{z}}, [\boldsymbol{z_0}]$) relies on an extra argument: an a priori, possibly not very tight, enclosure $[\boldsymbol{z_0}] \in \mathbb{IC}^k$ of the roots. This may be instantiated as the output of ValGcdRoot($[p], \boldsymbol{\ell}, \tilde{\boldsymbol{z}}$).

Algorithm 6 ValPejRoot($[p], \boldsymbol{\ell}, \tilde{\boldsymbol{z}}, [\boldsymbol{z_0}]$)

Input: $[p] \in \mathbb{IC}_n[x]$, $\boldsymbol{\ell} \in \mathbb{N}^k_{>0}$, root approximations $\tilde{\boldsymbol{z}} \in \mathbb{C}^k$, and initial root enclosures $[\boldsymbol{z_0}] \in \mathbb{IC}^k$

Output: refined root enclosures $[z] \in \mathbb{IC}^k$, or failure message

1: compute $A \approx J_{G_{\boldsymbol{\ell}}}(\tilde{\boldsymbol{z}})^+$ numerically
▷ *From now on, use interval arithmetic*
2: $[\boldsymbol{p}] \leftarrow \left(\frac{p_0}{p_n}, \ldots, \frac{p_{n-1}}{p_n}\right)$
3: $\boldsymbol{\delta} \leftarrow \mathrm{mag}\left([A]\ ([G_{\boldsymbol{\ell}}](\tilde{\boldsymbol{z}}) - [\boldsymbol{p}])\right)$
4: $\Lambda_0 \leftarrow \mathrm{mag}\left(I_k - [A]\,[J_{G_{\boldsymbol{\ell}}}]\,[\boldsymbol{z_0}]\right))$
5: ValidatedBounds($\Lambda_0, \boldsymbol{\delta}$) ▷ *raises an error if* $\rho(\Lambda_0) \geqslant 1$
6: $\Lambda : \boldsymbol{r} \in \mathbb{R}^k_{\geqslant 0} \mapsto \mathrm{mag}\left(I_k - [A]\,[J_{G_{\boldsymbol{\ell}}}]([\tilde{\boldsymbol{z}} \pm \boldsymbol{r}])\right)$
7: $\boldsymbol{r} \leftarrow$ ValidatedBounds($\Lambda, \boldsymbol{\delta}$)
8: $[\boldsymbol{z}] \leftarrow [\tilde{\boldsymbol{z}} \pm \boldsymbol{r}]$
9: **if** $[\boldsymbol{z}] \not\subseteq [\boldsymbol{z_0}]$ **then return** FAIL **end if**
10: **return** $[\boldsymbol{z}]$

Proposition 3.6. *Let* $[p] \in \mathbb{IC}_n[x]$, $\boldsymbol{\ell} \in \mathbb{N}^k_{>0}$, $\tilde{\boldsymbol{z}} \in \mathbb{C}^k$ *and* $[\boldsymbol{z_0}] \in \mathbb{IC}^k$. *Assume that for all* $p \in [p]$ *with multiplicity structure* $\boldsymbol{\ell}$, *there exists a unique vector of roots* $\boldsymbol{z} \in [\boldsymbol{z_0}]$ *such that* $p = p_n \prod_{i=1}^k (x - z_i)^{\ell_i}$. *Then if* ValPejRoot($[p], \boldsymbol{\ell}, \tilde{\boldsymbol{z}}, [\boldsymbol{z_0}]$) *does not fail, it computes refined root enclosures* $[\boldsymbol{z}] \subseteq [\boldsymbol{z_0}]$ *such that* $\boldsymbol{z} \in [\boldsymbol{z}]$.

Proof. The use of interval arithmetic after line 2 ensures that $\boldsymbol{\delta}$ is a rigorous upper bound for the defect $N_{G_{\boldsymbol{\ell}}}(\tilde{\boldsymbol{z}}) - \tilde{\boldsymbol{z}}$, and Λ_0 (resp. $\Lambda(\boldsymbol{r})$) a rigorous Lipschitz matrix for $N_{G_{\boldsymbol{\ell}}}$ over $[\boldsymbol{z_0}]$ (resp. $[\tilde{\boldsymbol{z}} \pm \boldsymbol{r}]$).

The first call to ValidatedBounds with the constant Lipschitz matrix $\boldsymbol{r} \mapsto \Lambda_0$ (the second argument, $\boldsymbol{\delta}$, is not relevant) ensures that $\rho(\Lambda_0) < 1$ by Proposition 3.3 and Theorem 3.2 (otherwise this routine raises an error). As a result, the initial root enclosure $[\boldsymbol{z_0}]$ contains *at most* one fixed point of $N_{G_{\boldsymbol{\ell}}}$.

The second call to ValidatedBounds, if it does not fail, returns an error vector $\boldsymbol{r} \in \mathbb{R}^k_{\geqslant 0}$ such that $N_{G_{\boldsymbol{\ell}}}$ contains a unique fixed point $\boldsymbol{z}^*$ inside $[\boldsymbol{z}] := [\tilde{\boldsymbol{z}} \pm \boldsymbol{r}]$. Also, if the inclusion $[\boldsymbol{z}] \subseteq [\boldsymbol{z_0}]$ in line 9 is satisfied, then it proves that this fixed point is also the unique fixed point of $N_{G_{\boldsymbol{\ell}}}$ in $[\boldsymbol{z_0}]$.

m	GcdRoot*	PejRoot	ValGcdRoot	ValPejRoot
4	1.25e-07	4.59e-15	5.62e-07	4.28e-13
5	4.99e-07	1.33e-14	1.05e-06	4.44e-13
6	4.01e-07	9.92e-15	2.08e-06	4.98e-13
7	4.35e-07	7.55e-15	2.89e-06	4.78e-13
8	6.47e-07	4.59e-15	4.80e-06	4.99e-13
9	1.09e-06	3.11e-15	6.74e-06	4.99e-13
10	1.13e-06	6.77e-15	8.92e-06	5.11e-13
11	1.83e-06	8.88e-16	1.22e-05	4.93e-13
12	9.01e-07	5.92e-16	1.61e-05	5.13e-13
13	3.29e-06	7.40e-15	2.00e-05	5.18e-13
14	2.52e-06	3.40e-15	2.62e-05	*(5.29e-13)*
15	6.33e-06	5.03e-15	3.14e-05	*(5.32e-13)*
16	1.24e-05	3.55e-15	—	*(5.26e-13)*
17	1.58e-05	7.85e-15	—	*(5.34e-13)*
18	1.36e-05	9.33e-15	—	*(5.31e-13)*
19	2.99e-05	1.33e-15	—	*(5.44e-13)*
20	4.91e-05	8.88e-16	—	*(5.37e-13)*

Table 2: Maximum relative errors on the root approximations and enclosures for p_m - Eq. (4). Results in parentheses for ValPejRoot indicate not fully rigorous enclosures due to the failure of the contraction test in line 5.

Now let $p \in [p]$ have multiplicity structure $\boldsymbol{\ell}$. By the assumption on $[z_0]$, there exists a unique $z \in [z_0]$ such that $p = p_n \prod (x - z_i)^{\ell_i}$. Clearly, z is a fixed point of $N_{G_{\boldsymbol{\ell}}}$. Hence, $z = z^* \in [z]$. □

Remark 3.7. Given $[p] \in \mathbb{IC}_n[x]$, Algorithms ValGcdRoot and ValPejRoot compute enclosures $[z] \in \mathbb{IC}^k$ for the roots of *any* $p \in [p]$ of multiplicity structure $\boldsymbol{\ell}$. They do however not prove that $[p]$ contains at least one such p. This can be done by checking that $p_n \prod_{i=1}^{k} (x - \tilde{z}_i) \in [p]$, with $\tilde{z}$ the computed numerical roots.

3.4 Validated Numerical Examples

Example 1 (validated version). We computed tight coefficientwise enclosures $[p_m]$ (up to machine double precision) for polynomials p_m of Example 1, and applied the full validation chain: from the root approximations $\tilde{z}$ computed by our routine GcdRoot* followed by Zeng's PejRoot, an initial enclosure $[z_0]$ is computed by ValGcdRoot and refined by ValPejRoot. The maximum relative errors obtained on the four roots are summarised in Table 2. We observe that the relative radius of $[z_0]$ grows with m roughly as fast as the relative error of the initial approximations computed by GcdRoot*. This follows from the growth with m of the condition number of Sylvester matrices, and hence of system (7). On the other hand, the growth with m of the condition number associated to system (3) is moderate, which explains why only two to three digits are lost in the intervals computed by ValPejRoot. However, for $m \geqslant 14$, ValPejRoot cannot fully certify its enclosures due to the failure of the contraction test in line 5: ValGcdRoot either fails to compute initial enclosures $[z_0]$, or they are too large. Nevertheless, these results are encouraging and show that the full validation chain solving Problem 2 on this example works for degrees up to 130 and multiplicities up to 52 using double precision only.

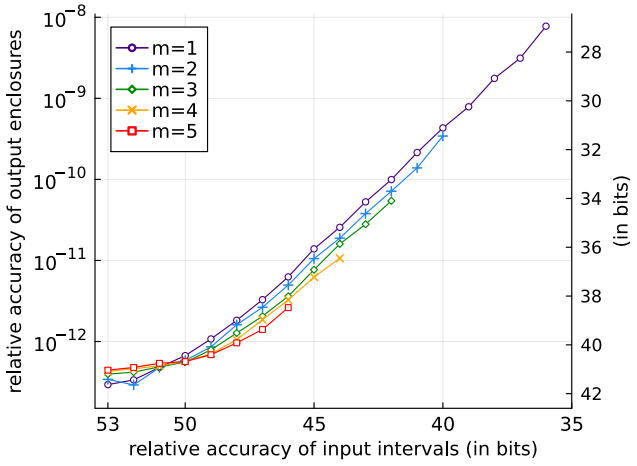

Figure 2: Accuracy of the computed root enclosures in function of accuracy of interval coefficients of $[p_m]$ - Eq. (4).

Example 1 (continued : the effect of uncertainties). For m up to 5, Figure 2 shows the relative accuracy of the root enclosures $[z]$ in function of the relative accuracy of the interval coefficients of $[p_m]$. First, we observe that the number of lost bits in the result is equal to the number of lost bits in the input *plus* a constant corresponding to the conditioning. This corroborates our claim that this validation method has *finite* conditioning: it is not subject to the "attainable accuracy barrier".

Unfortunately, the input accuracy threshold over which our method fails decreases with m, although the computed bounds themselves do not increase excessively. The first reason is the condition number of Sylvester matrices used by ValGcdRoot. Another source of failure we observed is the potentially very large overestimations in the interval evaluations of the maps $G_{\boldsymbol{\ell}}$ and $J_{G_{\boldsymbol{\ell}}}$ over non-thin intervals $[z]$, something well-known as the *wrapping effect* in interval analysis [21, §9.2].

Example 2 (nearby multiple roots). Figure 3 compares obtained accuracies of the approximations and validated enclosures for

$$(x - 1 + \varepsilon)^{m_1} (x - 1)^{m_2} (x + 0.5)^{m_3}, \tag{11}$$

already used as example in [28, §5.2]. The accuracy deteriorates when the gap ε between the two nearby multiple roots tends to 0. This is a consequence of the growth of the structure-preserving condition number when we get closer to the pejorative submanifold $\Pi_{m_1+m_2,m_3}$. On the other hand, the multiplicities do not seem to impact the accuracy significantly. The overestimation factor of the validated enclosures compared to the approximations is rather mild. However, higher multiplicities lower the threshold on ε over which the combination of ValGcdRoot and ValPejRoot fails.

Example 3 (complex roots). The following polynomial:

$$(x^2 - x + 1)^{m_1} (x^2 + 4x + 7)^{m_2} (x^2 - x - 1)^{m_3} (x^2 + 2x + 2)^{m_4} \tag{12}$$

has 8 complex roots depicted in Figure 4. The relative accuracies of the computed root approximations and enclosures for different values of the m_i are summarised in Table 3. It turns out that, in

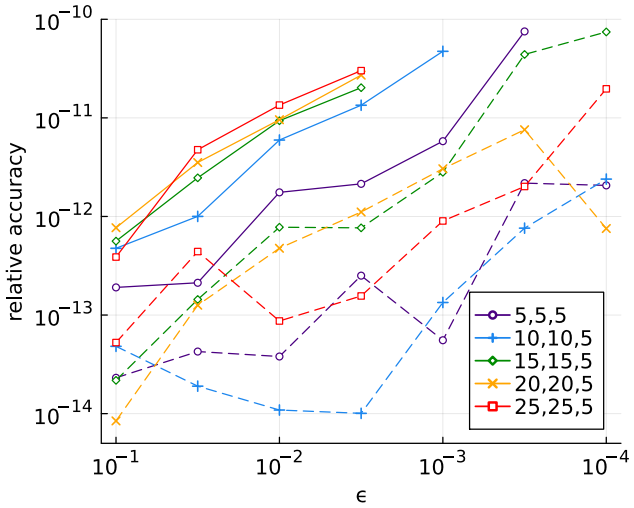

Figure 3: Relative accuracy of root approximations (dashed lines) and root enclosures (solid lines) for the polynomial (11) in function of ε and multiplicities (m_1, m_2, m_3).

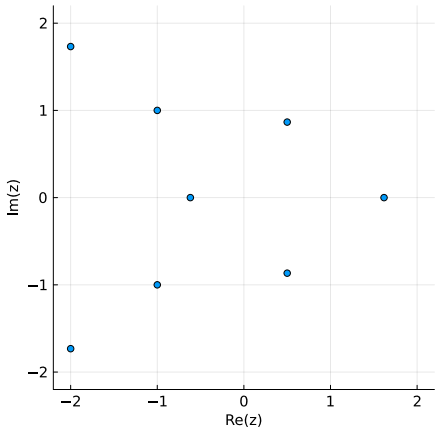

Figure 4: Complex roots of polynomial (12).

this example, the condition number of $J_{G_t}(z)$ grows significantly faster than the condition number of the Sylvester matrices $\Delta_k(p)$. As a result, the approximations produced by GcdRoot* are better than those of PejRoot (we thus kept the former for the validation), and ValGcdRoot performs better than ValPejRoot. Such cases are not discussed in [28, 30], and further research is needed to predict when they occur.

4 CONCLUDING REMARKS

In this article, we improved Zeng's numerical method [28, 30] to solve Problem 1 and proposed a validation method based on two algorithms ValGcdRoot and ValPejRoot to solve Problem 2.

A first future work is to refine our implementations of the validation routines so that their performances get closer to the purely numerical routines in terms of high multiplicities and obtained accuracy without using multi-precision. This, in particular, concerns Algorithm ValPejRoot where refined evaluation methods could lower the overapproximations on the defect and Lipschitz matrices, e.g., Discrete Fourier Transform (DFT) based evaluation methods to reduce the wrapping effect. Also, in cases where the pejorative condition number is large, one can consider combining the "one-root-at-a-time" deflation method of [22] with our two-stage validation approach to make it compliant with Problem 2.

On the theoretical side, a thorough complexity analysis of algorithms to solve Problem 2 will require a deeper analysis of the growth of the various condition numbers involved.

m_1, m_2, m_3, m_4	GcdRoot*	PejRoot	ValGcdRoot	ValPejRoot
2, 2, 1, 1	2.47e-15	3.98e-15	3.17e-13	6.75e-13
3, 1, 1, 3	4.87e-15	3.95e-14	6.01e-12	8.67e-11
5, 3, 3, 1	4.39e-15	6.83e-14	4.56e-13	5.12e-10
6, 3, 3, 1	2.02e-14	4.22e-14	7.97e-13	2.30e-09
2, 5, 5, 4	5.02e-13	1.16e-12	5.52e-11	—
8, 5, 2, 1	6.95e-15	1.59e-12	6.01e-13	—
8, 10, 3, 7	2.91e-10	8.12e-10	6.77e-09	—
20, 11, 7, 5	1.25e-13	1.26e-04	3.13e-12	—
25, 30, 3, 5	6.42e-10	1.83e-02	3.04e-08	—
25, 30, 3, 10	1.15e-07	8.04e-03	—	—

Table 3: Maximum relative errors on the root approximations and enclosures for the polynomials (12).

Acknowledgments. The authors are grateful to Bruno Salvy for useful discussions on detecting and converging to clusters of roots; to John Verzani for his kind help in using the `Polynomials.jl` package; and to the anonymous referees for relevant suggestions.

REFERENCES

[1] Bernhard Beckermann and George Labahn. 1998. When are Two Numerical Polynomials Relatively Prime? *Journal of Symbolic Computation* 26, 6 (1998), 677–689. https://doi.org/10.1006/jsco.1998.0234

[2] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. 2017. Julia: A fresh approach to numerical computing. *SIAM review* 59, 1 (2017), 65–98. https://doi.org/10.1137/141000671

[3] Dario A. Bini and Paola Boito. 2007. Structured Matrix-Based Methods for Polynomial ε-Gcd: Analysis and Comparisons. In *Proceedings of the 2007 International Symposium on Symbolic and Algebraic Computation (ISSAC '07)*. Association for Computing Machinery, New York, NY, USA, 9–16. https://doi.org/10.1145/1277548.1277551

[4] Paola Boito. 2012. *Structured matrix based methods for approximate polynomial GCD*. Vol. 15. Springer Science & Business Media. https://doi.org/10.1007/978-88-7642-381-9

[5] Robert M. Corless, Patrizia M. Gianni, Barry M. Trager, and Stephen M. Watt. 1995. The Singular Value Decomposition for Polynomial Systems. In *Proceedings of the 1995 International Symposium on Symbolic and Algebraic Computation (ISSAC '95)*. ACM, 195–207. https://doi.org/10.1145/220346.220371

[6] Ioannis Z. Emiris, André Galligo, and Henri Lombardi. 1997. Certified approximate univariate GCDs. *Journal of Pure and Applied Algebra* 117-118 (1997), 229–251. https://doi.org/10.1016/S0022-4049(97)00013-3

[7] Marc Giusti, Grégoire Lecerf, Bruno Salvy, and Jean-Claude Yakoubsohn. 2005. On location and approximation of clusters of zeros of analytic functions. *Foundations of Computational Mathematics* 5 (2005), 257–311. https://doi.org/10.1007/s10208-004-0144-z

[8] Gene H. Golub and Charles F. Van Loan. 1996. *Matrix computations* (3rd ed.). Johns Hopkins University Press. https://doi.org/10.56021/9781421407944

[9] Nicholas J. Higham. 2002. *Accuracy and Stability of Numerical Algorithms* (second ed.). Society for Industrial and Applied Mathematics. https://doi.org/10.1137/1.9780898718027

[10] Fredrik Johansson. 2017. Arb: Efficient Arbitrary-Precision Midpoint-Radius Interval Arithmetic. *IEEE Trans. Comput.* 66, 8 (2017), 1281–1292. https://doi.

org/10.1109/TC.2017.2690633
[11] William Kahan. 1972. *Conserving confluence curbs ill-condition.* Technical Report. https://apps.dtic.mil/sti/citations/AD0766916
[12] Angelos Mantzaflaris and Bernard Mourrain. 2011. Deflation and Certified Isolation of Singular Zeros of Polynomial Systems. In *Proceedings of the 36th International Symposium on Symbolic and Algebraic Computation* (San Jose, California, USA) *(ISSAC '11)*. ACM, 249–256. https://doi.org/10.1145/1993886.1993925
[13] Ramon E. Moore, R. Baker Kearfott, and Michael J. Cloud. 2009. *Introduction to Interval Analysis.* Society for Industrial and Applied Mathematics. https://doi.org/10.1137/1.9780898717716
[14] C.Andrew Neff and John H. Reif. 1996. An Efficient Algorithm for the Complex Roots Problem. *Journal of Complexity* 12, 2 (1996), 81–115. https://doi.org/10.1006/jcom.1996.0008
[15] Arnold Neumaier. 1991. *Interval Methods for Systems of Equations.* Cambridge University Press. https://doi.org/10.1017/CBO9780511526473
[16] Victor Y. Pan. 1997. Solving a Polynomial Equation: Some History and Recent Progress. *SIAM Rev.* 39, 2 (1997), 187–220. https://doi.org/10.1137/S0036144595288554
[17] Victor Y. Pan. 2002. Univariate Polynomials: Nearly Optimal Algorithms for Numerical Factorization and Root-finding. *Journal of Symbolic Computation* 33, 5 (2002), 701–733. https://doi.org/10.1006/jsco.2002.0531
[18] Jiří Rohn. 1996. Enclosing solutions of overdetermined systems of linear interval equations. *Reliable Computing* 2 (1996), 167–171. https://doi.org/10.1007/BF02425920
[19] Siegfried M. Rump. 1999. *INTLAB—INTerval LABoratory.* Springer. https://doi.org/10.1007/978-94-017-1247-7_7
[20] Siegfried M. Rump. 2003. Ten methods to bound multiple roots of polynomials. *J. Comput. Appl. Math.* 156, 2 (2003), 403–432. https://doi.org/10.1016/S0377-0427(03)00381-9
[21] Siegfried M. Rump. 2010. Verification methods: Rigorous results using floating-point arithmetic. *Acta Numerica* 19 (2010), 287–449. https://doi.org/10.1017/S096249291000005X
[22] Siegfried M. Rump and Stef Graillat. 2010. Verified error bounds for multiple roots of systems of nonlinear equations. *Numerical Algorithms* 54 (2010), 359–377. https://doi.org/10.1007/s11075-009-9339-3
[23] Tateaki Sasaki and Matu-Tarow Noda. 1989. Approximate square-free decomposition and root-finding of ill-conditioned algebraic equations. *J. Inform. Process* 12, 2 (1989), 159–168.
[24] Arnold Schönhage. 1985. Quasi-GCD computations. *Journal of Complexity* 1, 1 (1985), 118–137. https://doi.org/10.1016/0885-064X(85)90024-X
[25] Hans J Stetter. 2004. *Numerical polynomial algebra.* SIAM.
[26] Lloyd N Trefethen and David Bau. 1997. *Numerical Linear Algebra.* Vol. 181. Society for Industrial and Applied Mathematics.
[27] Warwick Tucker. 2011. Validated Numerics: A Short Introduction to Rigorous Computations. Princeton University Press. https://doi.org/10.2307/j.ctvcm4g18
[28] Zhonggang Zeng. 2003. A Method Computing Multiple Roots of Inexact Polynomials. In *Proceedings of the 2003 International Symposium on Symbolic and Algebraic Computation (ISSAC '03)*. ACM, 266–272. https://doi.org/10.1145/860854.860907
[29] Zhonggang Zeng. 2004. Algorithm 835: MultRoot—a Matlab Package for Computing Polynomial Roots and Multiplicities. *ACM Trans. Math. Softw.* 30, 2 (jun 2004), 218–236. https://doi.org/10.1145/992200.992209
[30] Zhonggang Zeng. 2005. Computing multiple roots of inexact polynomials. *Math. Comp.* 74, 250 (2005), 869–903. https://doi.org/10.1090/S0025-5718-04-01692-8
[31] Zhonggang Zeng and Barry H. Dayton. 2004. The Approximate GCD of Inexact Polynomials. In *Proceedings of the 2004 International Symposium on Symbolic and Algebraic Computation (ISSAC '04)*. ACM, 320–327. https://doi.org/10.1145/1005285.1005331