New practical advances in polynomial root clustering

Rémi Imbach[1] * and Victor Y. Pan[2] **

[1] Courant Institute of Mathematical Sciences
New York University, USA
Email: remi.imbach@nyu.edu
https://cims.nyu.edu/~imbach/
[2] City University of New York, USA
Email: victor.pan@lehman.cuny.edu
http://comet.lehman.cuny.edu/vpan/

Abstract. We report an ongoing work on clustering algorithms for complex roots of a univariate polynomial p of degree d with real or complex coefficients. As in their previous best subdivision algorithms our root-finders are robust even for multiple roots of a polynomial given by a black box for the approximation of its coefficients, and their complexity decreases at least proportionally to the number of roots in a region of interest (ROI) on the complex plane, such as a disc or a square, but we greatly strengthen the main ingredient of the previous algorithms. Namely our new counting test essentially amounts to the evaluation of a polynomial p and its derivative p', which is a major benefit, e.g., for sparse polynomials p. Moreover with evaluation at about $\log(d)$ points (versus the previous record of order d) we output correct number of roots in a disc whose contour has no roots of p nearby. Moreover we greatly soften the latter requirement versus the known subdivision algorithms. Our second and less significant contribution concerns subdivision algorithms for polynomials with real coefficients. Our tests demonstrate the power of the proposed algorithms.

1 Introduction

We seek complex roots of a degree d univariate polynomial p with real or complex coefficients. For a while the user choice for this problem has been (the package `MPsolve`) based on *e.g.* Erhlich-Aberth (simultaneous Newton-like) iterations. Their empirical global convergence (right from the start) is very fast, but its formal support is a long-known challenge, and the iterations approximate the roots in a fixed region of interest (ROI) about as slow as all complex roots.

In contrast, for the known algorithms subdividing a ROI, e.g., box, the cost of root-finding in a ROI decreases at least proportionally to the number of roots in

* Rémi's work is supported by NSF Grants # CCF-1563942 and # CCF-1708884.
** Victor's work is supported by NSF Grants # CCF-1116736 and # CCF-1563942 and by PSC CUNY Award 698130048.

it. Some recent subdivision algorithms have a proved nearly optimal complexity, are robust in the case of root clusters and multiple roots, and their implementation in [IPY18] a little outperforms `MPsolve` for ROI containing only a small number of roots, which is an important benefit in many computational areas.

The Local Clustering Problem For a complex set $\mathcal{S}$, $\texttt{Zero}(\mathcal{S}, p)$, or sometimes $\texttt{Zero}(\mathcal{S})$, stands for the roots of p in $\mathcal{S}$. $\#(\mathcal{S}, p)$ (or $\#(\mathcal{S})$) stands for the number of roots of p in $\mathcal{S}$. Here and hereafter the roots are counted with their multiplicity.

We consider boxes (that is, squares with horizontal and vertical edges, parallel to coordinate axis) and discs $D(c, r) = \{z \text{ s.t. } |z - c| \leq r\}$ on the complex plane. For such a box (resp. disc) $\mathcal{S}$ and a positive δ we denote by $\delta\mathcal{S}$ its concentric δ-dilation. We call a disc Δ an *isolator* if $\#(\Delta) > 0$ and call it *natural* isolator if in addition $\#(\Delta) = \#(3\Delta)$. A set $\mathcal{R}$ of roots of p is called a *natural cluster* if there exists a natural isolator Δ with $\texttt{Zero}(\mathcal{R}) = \texttt{Zero}(\Delta)$. The Local Clustering Problem (LCP) is to compute natural isolators for natural clusters together with the sum of multiplicities of roots in the clusters:

Local Clustering Problem (LCP):
Given: a polynomial $p \in \mathbb{C}[z]$, a ROI $B_0 \subset \mathbb{C}$, $\epsilon > 0$
Output: a set of pairs $\{(\Delta^1, m^1), \ldots, (\Delta^\ell, m^\ell)\}$ where:
- the Δ^j's are pairwise disjoint discs of radius $\leq \epsilon$,
- $m^j = \#(\Delta^j, p) = \#(3\Delta^j, p)$ and $m^j > 0$ for $j = 1, \ldots, \ell$
- $\texttt{Zero}(B_0, p) \subseteq \bigcup_{j=1}^{\ell} \texttt{Zero}(\Delta^j, p) \subseteq \texttt{Zero}(2B_0, p)$.

The basic tool of the nearly optimal subdivision algorithm of [BSS+16] for the LCP is the *T^*-test* for counting the roots of p in a complex disc (with multiplicity). It relies on Pellet's theorem, involves approximations of the coefficients of p, and applies shifting and scaling the variable z and Dandelin-Gräffe's root-squaring iterations. [IPY18] describes high-level improvement of this test, and `Ccluster`[3], a `C` implementation of [BSS+16].

Our contributions Our new counting test, the *P^*-test*, for a pair of complex c and positive r computes the number s_0 of roots of p in a complex disc Δ centered at c with radius r. If the boundary $\partial\Delta$ contains no roots of p, then

$$s_0 = \frac{1}{2\pi\mathbf{i}} \int_{\partial\Delta} \frac{p'(z)}{p(z)} dz, \text{ for } \mathbf{i} = \sqrt{-1}, \tag{1}$$

by virtue of Cauchy's theorem. By following [Sch82] and [Pan18], we approximate s_0 by s_0^* obtained by evaluating p'/p on q points on the boundary $\partial\Delta$ within the error bound $|s_0 - s_0^*|$ in terms of q and the relative width of a root-free annulus around $\partial\Delta$. Namely if $\#(\frac{1}{2}\Delta) = \#(2\Delta)$ then for $q = \lceil \log_2(d+4) + 2 \rceil$ we recover exact value of s_0 from s_0^*.

[3] https://github.com/rimbach/Ccluster

	Ccluster local				Ccluster global				MPsolve
	#Clus	t_{old}	t_{new}	t_{old}/t_{new}	#Clus	t_{old}	t_{new}	t_{old}/t_{new}	t
$\texttt{Mignotte}_{128}$	1	0.05	0.02	2.49	127	5.00	1.81	2.75	0.02
$\texttt{Mignotte}_{256}$	1	0.16	0.05	2.82	255	31.8	10.7	2.95	0.07
$\texttt{Mignotte}_{383}$	1	0.32	0.11	2.74	382	79.7	26.8	2.97	0.17
$\texttt{Mandelbrot}_{7}$	1	0.18	0.06	2.92	127	7.17	2.88	2.48	0.06
$\texttt{Mandelbrot}_{8}$	0	0.39	0.11	3.38	255	40.6	15.1	2.69	0.39
$\texttt{Mandelbrot}_{9}$	5	3.08	0.91	3.37	511	266	97.1	2.74	3.20

Table 1. Running times in seconds of `Ccluster`, new and old versions, for computing clusters of roots in a small ROI (local) and a ROI containing all the roots, and `MPsolve`.

We give an *effective*[4] (*i.e.* implementable) description of this P^*-test, which involves no coefficients of p and can be applied to a polynomial p represented by a black box for its evaluation. For sparse polynomials and polynomials defined by recursive process such as Mandelbrot's polynomials (see [BF00], or Eq. (3) below), the test is particularly efficient and the resulting acceleration of the clustering algorithm of [BSS+16] is particularly strong.

Our second (and less significant) contribution applies to polynomials with real coefficients: the roots of such polynomials are either real or appear in complex conjugated pairs. As a consequence, one can recover all the roots in a ROI B_0 containing $\mathbb{R}$ from the ones with positive imaginary parts. We show how to improve a subdivision scheme by leveraging of the latter property.

Every polynomial p and its product $p\overline{p}$ with its complex conjugate $\overline{p}$ belongs to this class and has additional property that the multiplicity of its real roots is even, but we do not assume the latter restriction.

We implemented and tested our improvements in `Ccluster`. For polynomials with real coefficients that are sparse or can be evaluated by a fast procedure, we achieved a 2.5 to 3 fold speed-up as shown in table 1 by columns t_{old}/t_{new}. When the ROI contains only a few solutions, `Ccluster` is, thanks to those improvements, a little more efficient than `MPsolve` (compare columns `Ccluster` local, t_{new} and `MPsolve` in table 1). We give details on our experiments below.

Implementation and experiments. All the timings shown in this article are sequential times in seconds on a Intel(R) Core(TM) i7-7600U CPU @ 2.80GHz machine with Linux. `MPsolve` is called with the command `mpsolve -as -Gi -o16 -j1`[5]. Table 1 shows comparative running times of `Ccluster` and `MPsolve` on two families of polynomials, Mignotte and Mandelbrot's polynomials, with real coefficients, defined below. Columns t_{new} (resp. t_{old}) show timings of `Ccluster` with (resp. without) the improvements described in this paper. Columns #Clus show the number of clusters found by two versions. We used both versions of `Ccluster` with $\epsilon = 2^{-53}$. `Ccluster` global refers to the ROI $[-500, 500] + \mathbf{i}[-500, 500]$, that contains all the roots of the tested polynomials; `Ccluster` local refers to an ROI containing only a few solutions. We used $[-0.5, 0.5]+\mathbf{i}[-0.5, 0.5]$ for Mignotte's polynomials and $[-0.25, 0.25] + \mathbf{i}[-0.25, 0.25]$ for Mandelbrot's polynomials.

[4] by effective, we refer to the pathway proposed in [XY19] to describe algorithms in three levels: abstract, interval, effective

[5] `MPsolve` tries to isolate the roots unless the escape bound 10^{-16} is reached.

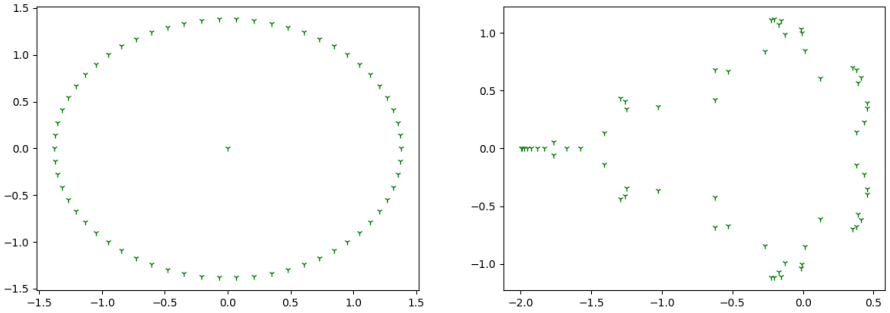

Fig. 1. Left: 63 clusters of roots for a Mignotte polynomial of degree 64. **Right:** Clusters of roots for the Mandelbrot polynomial of degree 63.

The Mignotte's polynomial of degree d and parameter $a = 14$ is:

$$\texttt{Mignotte}_d(z) = z^d - 2(2^a z - 1)^2 \tag{2}$$

It has a cluster of two roots near the origin whose separation is near the theoretical minimum separation bound. It is sparse and can be evaluated very fast. We define the Mandelbrot's polynomial as $\texttt{Mandelbrot}_1(z) = 1$ and

$$\texttt{Mandelbrot}_k(z) = z\texttt{Mandelbrot}_{k-1}(z)^2 + 1 \tag{3}$$

$\texttt{Mandelbrot}_k(z)$ has degree $2^k - 1$. It can be evaluated with a straight line program. The 63 clusters of roots of $\texttt{Mandelbrot}_6(z)$ and $\texttt{Mignotte}_{64}(z)$ are depicted in Fig. 1.

Structure of the paper. Our paper is organized as follows: in Sec. 2 we describe our P^*-test. In Sec. 3 we apply it to speeding up a clustering algorithm. In Sec. 4 we cover our root-finder for polynomials with real coefficients. Sec. 5 presents the results of our improvements. In the rest of the present section, we recall the related work and the clustering algorithm of [BSS+16].

1.1 Previous works

Univariate polynomial root-finding is a long-standing and still actual problem; it is intrinsically linked to numerical factorization of a polynomial into the product of its linear factors. The algorithms of [Pan02] support record and nearly optimal bounds on the Boolean complexity of the solution of both problems of factorization and root-finding. The cost bound of the factorization is smaller by a factor of d, and both bounds differ from respective information lower bound by at most a polylogarithmic factor in the input size and in the bound on the required output precision. Root-finder supporting such bit complexity bounds are said to be nearly optimal. The algorithms of [Pan02] are involved and have never been implemented. User's choice has been for a while the package of subroutines `MPsolve` (see [BF00] and [BR14]), based on simultaneous Newton-like

(*i.e.* Ehrlich-Aberth) iterations. These iterations converge to all roots simultaneously with cubic convergence rate, but only locally, that is, near the roots; empirically they converge very fast also globally, right from the start, although formal support for this empirical behavior is a long-known research challenge. Furthermore these iterations compute a small number of roots in a ROI not much faster than all roots.

In contrast, recent approaches based on subdivision (as well as the algorithms of [Pan02]) compute the roots in a fixed ROI at a cost that decreases at least proportionally to the number of roots. Near-optimal complexity has been achieved both for the real case (see [PT16], [SM16] that combines the Descartes rule of signs with Newton's iterations and its implementation in [KRS16]) and the complex case. In the complex case [BSSY18] similarly combines counting test based on Pellet's theorem with complex version of the QIR algorithm, which in turn combines Newton's and secant iterations.

[BSS+16] extends the method of [BSSY18] for root clustering, *i.e.* it solves the LCP and is robust in the case of multiple roots; its implementation ([IPY18]) is a little more efficient than `MPsolve` for ROI's containing only several roots; when all the roots are sought, `MPsolve` remains the user's choice. The algorithms of [BSS+16] and [BSSY18] are direct successors of the previous subdivision algorithms of [Ren87] and [Pan00], presented under the name of Quad-tree algorithms (inherited from the earlier works by Henrici and Gargantini).

Besides Pellet's theorem, counting test in ROI can rely on Eq. (1) and winding numbers algorithms (see, e.g., [HG69,Ren87] and [ZZ19]).

1.2 Solving the LCP

C^0 and C^* tests The two tests C^0 and C^* discard boxes with no roots of p and count the number of roots in a box, respectively. For a given complex disc Δ, $C^0(\Delta, p)$ returns either -1 or 0, and returns 0 only if p has no root in Δ, while $C^*(\Delta, p)$ returns an integer $k \geq -1$ such that $k \geq 0$ only if p has k roots in Δ. Below, we may write $C^0(\Delta)$ for $C^0(\Delta, p)$ and $C^*(\Delta)$ for $C^*(\Delta, p)$.

In [BSS+16,BSSY18,IPY18], both C^0 and C^* are based on the so called "soft Pellet test" denoted $T^*(\Delta, p)$ or $T^*(\Delta)$ which returns an integer $k \geq -1$ such that $k \geq 0$ only if p has k roots in Δ:

$$C^0(\Delta) := \begin{cases} 0 & \text{if } T^*(\Delta) = 0 \\ -1 & \text{otherwise} \end{cases}$$
$$C^*(\Delta) := T^*(\Delta). \tag{4}$$

Boxes, quadri-section and connected components The box B centered in $c = a + \mathbf{i}b$ with width w is defined as $[a - w/2, a + w/2] + \mathbf{i}[b - w/2, b + w/2]$. We denote by $w(B)$ the width of B. We call *containing disc* of B the disc $\Delta(B)$ defined as $D(c, \frac{3}{4}w(B))$. We define the four children of B as the four boxes centered in $(a \pm \frac{w}{4}) + \mathbf{i}(b \pm \frac{w}{4})$ with width $\frac{w}{2}$.

Algorithm 1 Root Clustering Algorithm

Input: A polynomial $p \in \mathbb{C}[z]$, a ROI B_0, $\epsilon > 0$; suppose p has no roots in $2B_0 \setminus B_0$
Output: Set R of components solving the LCP.

$R \leftarrow \emptyset$, $Q \leftarrow \{B_0\}$ *// Initialization*
while Q is not empty **do** *// Main loop*
  $\mathcal{C} \leftarrow Q.pop()$ *//$\mathcal{C}$ has the widest containing box in Q*
  // Validation
  if $w(\mathcal{C}) \leq \epsilon$ **and** $\mathcal{C}$ is compact **and** $\mathcal{C}$ is separated from Q **then**
    $k \leftarrow C^*(\Delta(\mathcal{C}), p)$
    if $k > 0$ **then**
      $R.push((\mathcal{C}, k))$
      break
  // Bisection
  $S \leftarrow$ empty set of boxes
  for each box B of $\mathcal{C}$ **do**
    for each child B' of B **do**
      if $C^0(\Delta(B'), p)$ returns -1 **then**
        $S.push(B')$
  $Q.push($ connected components in S $)$
return R

Recursive subdivisions of a ROI B_0 falls back to the construction of a tree rooted in B_0. Hereafter we refer to boxes that are nodes (and possibly leafs) of this tree as the boxes of the subdivision tree of B_0.

A *component* $\mathcal{C}$ is a set of connected boxes. The component box $B_{\mathcal{C}}$ of a component $\mathcal{C}$ is a smallest square box subject to $\mathcal{C} \subseteq B_{\mathcal{C}} \subseteq B_0$, where B_0 is the initial ROI. We write $\Delta(\mathcal{C})$ for $\Delta(B_{\mathcal{C}})$ and $w(\mathcal{C})$ for $w(B_{\mathcal{C}})$. Below we consider components made up of boxes of the same width; such a component is *compact* if $w(\mathcal{C})$ is at most 3 times the width of its boxes. Finally, a component $\mathcal{C}$ is *separated* from a set S if $\forall \mathcal{C}' \in S, 4\Delta(\mathcal{C}) \cap \mathcal{C}' = \emptyset$ and $4\Delta(\mathcal{C}) \subseteq 2B_0$.

A root clustering algorithm We give in Algo. 1 a simple root clustering algorithm based on subdivision of ROI B_0. For convenience we assume that p has no root in $2B_0 \setminus B_0$ but this limitation can easily be removed. The paper [BSS+16] proves that Algo. 1 terminates and output correct solution provided that the C^0 and C^*-tests are as in Eq. (4).

Note that in the **while** loop of Algo. 1, components with widest containing box are processed first; together with the definition of a separated component, this implies the following remark:

Remark 1 *Let $\mathcal{C}$ be a component in Algo. 1 that passes the test in step 4. Then $\mathcal{C}$ satisfies $\#(\Delta(\mathcal{C})) = \#(4\Delta(\mathcal{C}))$.*

2 Counting the number of roots in a well isolated disk

In this section we present a new test for counting the number of roots with multiplicity of p in a disc Δ provided that the roots in Δ are well isolated from the other roots of p. Let us first formalize this notion:

Definition 2 (Isolation ratio) *A complex disc Δ has isolation ratio ρ for a polynomial p if $\rho > 1$ and $\texttt{Zero}(\frac{1}{\rho}\Delta) = \texttt{Zero}(\rho\Delta)$.*

Let $\texttt{Zero}(\Delta) = \{\alpha_1, \ldots, \alpha_{d_\Delta}\}$ and let m_i be the multiplicity of α_i. The h-th power sum of the roots in Δ is the complex number

$$s_h = \sum_{i=1}^{d_\Delta} m_i \times \alpha_i^h \tag{5}$$

In our test, called hereafter P^*-test, we approximate the 0-th power sum s_0 of the roots of p in Δ equal to the number of roots of p in Δ (counted with multiplicity). We obtain precise s_0 from s_0^* where p and its derivative p' are evaluated on only a small number of points on the contour of Δ. For instance, if Δ has isolation ratio 2 and p has degree 500, our test amounts to evaluating p and p' on $q = 11$ points; s_0 is recovered from these values in $O(q)$ arithmetic operations.

If p and its derivative can be evaluated at a low computational cost, e.g. when p is sparse or p is defined by a recurrence as the Mandelbrot polynomial (see [BF00][Eq. (16)] or Eq. (3) above), our P^*-test can be substantially cheaper to apply than the T^*-test presented above. Notice however that it requires the isolation ratio of Δ (or at least a lower bound) to be known.

2.1 Approximation of the 0-th power sum of the roots in a disk

[Sch82] and [Pan18] give formulas for approximating the powers sums s_h of the roots in the unit disk. Here we compute s_0 in any complex disk $\Delta = D(c, r)$.

For a positive integer q, define

$$s_0^* = \frac{r}{q} \sum_{g=0}^{q-1} \omega^g \frac{p'(c + r\omega^g)}{p(c + r\omega^g)} \tag{6}$$

where $\omega = e^{\frac{2\pi \mathrm{i}}{q}}$ denotes a primitive q-th root of unity.

The theorem below shows that the latter number approximates the 0-th power sum with an error that can be made as tight as desired by increasing q, providing that Δ has isolation ratio noticeably exceeding 1.

Theorem 3 *Let Δ have isolation ratio ρ for p, let $\theta = 1/\rho$, let s_0 be the 0-th power sum of the roots of p in Δ, and let s_0^* be defined as in eq. 6. Then*

(i) $|s_0^* - s_0| \leq \dfrac{d\theta^q}{1 - \theta^q}$.

(ii) Fix $e > 0$. If $q = \lceil \log_\theta(\frac{e}{d+e}) \rceil$ then $|s_0^* - s_0| \leq e$.

Proof of Thm. 3: Let $p_\Delta(z)$ be the polynomial $p(c+rz)$. Thus $p'_\Delta(z) = rp'(c+rz)$ and Eq. (6) rewrites $s_0^* = \frac{1}{q}\sum_{g=0}^{q-1} \omega^g \frac{p'_\Delta(\omega^g)}{p_\Delta(\omega^g)}$. In addition, the unit disk $D(0,1)$ has isolation ratio ρ for p_Δ and contains s_0 roots of p_Δ. Then apply Thm. 14 in [Pan18] to $p_\Delta(z)$ to obtain (i). (ii) is a direct consequence of (i). □

For example, if Δ has isolation ratio 2, p has degree 500 and one wants to approximate s_0 with an error less than $1/4$, it suffices to apply formula in Eq. (6) for $q = 11$, that is to evaluate p and its derivative p' at 11 points.

2.2 Black box for evaluating a polynomial p on an oracle number

Our goal is to give an effective description of our P^*-test; to this end, let us introduce the notion of *oracle numbers* that correspond to black boxes giving arbitrary precision approximations of any complex number. Such oracle numbers can be implemented through arbitrary precision interval arithmetic or ball arithmetic. Let $\square\mathbb{C}$ be the set of complex intervals. If $\square a \in \square\mathbb{C}$, then $w(\square a)$ is the maximum width of real and imaginary parts of $\square a$.

For a number $a \in \mathbb{C}$, we call *oracle* for a a function $\mathcal{O}_a : \mathbb{N} \to \square\mathbb{C}$ such that $a \in \mathcal{O}_a(L)$ and $w(\mathcal{O}_a(L)) \leq 2^{-L}$ for any L. Let $\mathcal{O}_\mathbb{C}$ be the set of oracle numbers.

For a polynomial $p \in \mathbb{C}[z]$, we call *evaluation oracle* for p a function $\mathcal{I}_p : (\mathcal{O}_\mathbb{C}, \mathbb{N}) \to \square\mathbb{C}$, such that if $\mathcal{O}_a$ is an oracle for a and $L \in \mathbb{N}$, then $p(a) \in \mathcal{I}_p(\mathcal{O}_a, L)$ and $w(\mathcal{I}_p(\mathcal{O}_a, L)) \leq 2^{-L}$.

We consider evaluation oracles $\mathcal{I}_p$ and $\mathcal{I}_{p'}$ for p and its derivative p'. If p is given by $d+1$ oracles for its coefficients, one can easily construct $\mathcal{I}_p$ and $\mathcal{I}_{p'}$ by using for instance Horner's rule. However for some polynomials defined by a procedure, for instance the Mandelbrot polynomial (see Eq. (3)), one can construct fast evaluation oracles $\mathcal{I}_p$ and $\mathcal{I}_{p'}$ from the procedurial definition.

2.3 The P^*-test

Algo. 2 counts the number of roots of p in a disk $\Delta = D(c,r)$ having isolation ratio at least ρ. For such a disk, any positive integer q and any integer $0 \leq g < q$, one has $p(c+r\omega^g) \neq 0$. As a consequence, there exist an L' s.t $\forall L \geq L', \forall 0 \leq g \leq q-1, 0 \notin \mathcal{I}_p(\mathcal{O}_{c+r\omega^g}, L)$ and the intervals $\square s_0^*$ computed in step 4 of Algo. 2 have strictly decreasing width as of $L \geq L'$. This shows the termination of Algo. 2. Its correctness is stated in the following proposition:

Proposition 4 *Let k be the result of the call $P^*(\mathcal{I}_p, \mathcal{I}_{p'}, \Delta, \rho)$. If Δ has isolation ratio at least ρ for p, then p has k roots in Δ counted with multiplicity.*

Proof of Prop. 4. Once the **while** loop in Algo. 2 terminates, the interval $\square s_0^*$ contains s_0^* and $w(\square s_0^*) < 1/2$. In addition, by virtue of statement (ii) of Thm. 3, one has $|s_0^* - s_0| \leq 1/4$, thus $\square s_0$ defined in step 7 satisfies: $w(\square s_0) < 1$

Algorithm 2 $P^*(\mathcal{I}_p, \mathcal{I}_{p'}, \Delta, \rho)$

Input: $\mathcal{I}_p$, $\mathcal{I}_{p'}$ evaluation oracles for p and p', $\Delta = D(c, r)$, $\rho > 1$. p has degree d.
Output: an integer in $\{0, \ldots, d\}$

1: $L \leftarrow 53$, $w \leftarrow 1$, $e \leftarrow 1/4$, $\theta \leftarrow 1/\rho$
2: $q \leftarrow \lceil \log_\theta(\frac{e}{d+e}) \rceil$
3: **while** $w \geq 1/2$ **do**
4: Compute interval $\square s_0^*$ as $\frac{r}{q} \sum_{g=0}^{q-1} \mathcal{O}_{\omega^g}(L) \frac{\mathcal{I}_{p'}(\mathcal{O}_{c+r\omega^g}, L)}{\mathcal{I}_p(\mathcal{O}_{c+r\omega^g}, L)}$
5: $w \leftarrow w(\square s_0^*)$
6: $L \leftarrow 2 * L$
7: $\square s_0 \leftarrow \square s_0^* + [-1/4, 1/4] + \mathbf{i}[-1/4, 1/4]$
8: **return** the unique integer in $\square s_0$

and $s_0 \in \square s_0$. Since $\square s_0$ contains at most one integer, s_0 is the unique integer in $\square s_0$, and is equal to the number of roots in Δ.

□

3 Using the P^*-test in a subdivision framework

Let us discuss the use of the P^*-test as C^0 and C^*-tests in order to speed up Algo. 1. Table. 2 covers runs of Algo. 1 on Mignotte and Mandelbrot's polynomials. t is the running time when C^0 and C^* tests are defined by Eq. (4). Columns nb show the respective numbers of C^0 and C^*-tests performed, column t_0 and t_0/t (resp. t_* and t_*/t) show time and ratio of times spent in C^0 (resp. C^*) tests when it is defined by Eq. (4).

One can readily use the P^*-test to implement the C^*-test by defining

$$C^*(\Delta) := P^*(\mathcal{I}_p, \mathcal{I}_{p'}, 2\Delta, 2) \tag{7}$$

Following Rem. 1, the C^*-test is called in Algo. 1 for components $\mathcal{C}$ satisfying $\#(\Delta(\mathcal{C})) = \#(4\Delta(\mathcal{C}))$. Hence $2\Delta(\mathcal{C})$ has isolation ratio 2 and by virtue of Prop. 4, $C^*(\Delta(\mathcal{C}))$ returns $r \geq 0$ only if $\Delta(\mathcal{C})$ contains r roots.

However this would not imply much improvements in itself. Column t'_* in table. 2 shows the time that would be spent in C^*-tests if it was defined by Eq. (7): it is far less than t_*, but the ratio of time spent in C^*-tests (see column t_*/t) is very small. In contrast, about 90% of the running time of Algo. 1 is spent in C^0-tests (see column t_0/t). We propose to use a modified version of the P^*-test as a filter in the C^0-test to decrease its running time.

3.1 An approximate P^*-test

The approximate version of the P^*-test is aimed at being applied to a disk $\Delta = D(c, r)$ with unknown isolation ratio. Unless Δ has isolation ratio $\rho > 1$, the very unlikely case where for some $0 \leq g < q$, $p(c + r\omega^g) = 0$, leads to a non-terminating call of $P^*(\mathcal{I}_p, \mathcal{I}_{p'}, \Delta, \rho)$. Also, $\square s_0$ computed in step 7 of Algo. 2

	C^0-tests							C^*-tests			
	T^*-tests			$\widetilde{P^*}$-tests				T^*-tests			P^*-tests
	nb	t_0	t_0/t (%)	t'_0	n_{-1}	n_{-2}	n_{err}	nb	t_*	t_*/t (%)	t'_*
$\texttt{Mignotte}_{128}$	4508	4.73	90.9	0.25	276	0	12	128	0.07	1.46	0.01
$\texttt{Mignotte}_{256}$	8452	27.8	91.2	0.60	544	0	20	256	0.58	1.92	0.02
$\texttt{Mandelbrot}_7$	4548	6.34	88.1	0.28	168	0	28	131	0.11	1.51	0.01
$\texttt{Mandelbrot}_8$	8892	35.6	88.4	0.67	318	0	57	256	0.69	1.71	0.03

Table 2. Details on runs of Algo. 1 on Mignotte and Mandelbrot's polynomials.

Algorithm 3 $\widetilde{P^*}(\mathcal{I}_p, \mathcal{I}_{p'}, \Delta, \rho)$

Input: $\mathcal{I}_p$, $\mathcal{I}_{p'}$ evaluation oracles for p and p', $\Delta = D(c,r)$, $\rho > 0$. p has degree d.
Output: an integer in $\{-2, -1, 0, \ldots, d\}$

could contain no integer or an integer that is not s_0. We define the $\widetilde{P^*}$-test specified in Algo. 3 by modifying Algo. 2 as follows:

1. after step 3, if an $\mathcal{I}_p(\mathcal{O}_{c+r\omega^g}, L)$ contains 0, the result -2 is returned;
2. step 7 is replaced with: $\square s_0 \leftarrow \square s_0^* + [-1/2, 1/2] + \mathbf{i}[-1/2, 1/2]$,
3. after step 7, unless $\square s_0$ contains a unique integer, the result -1 is returned.

Modification 1 ensures termination when Δ does not have isolation ratio $\rho > 1$. With modification 2, $\square s_0$ can have width greater than 1 and contain more than one integer. With modification 3, the $\widetilde{P^*}$-test can return -1 which means that no conclusion can be made. If $\widetilde{P^*}(\mathcal{I}_p, \mathcal{I}_{p'}, \Delta, \rho)$ returns a positive integer, this result has still to be checked, for instance, with the T^*-test.

In table 2, column n_{-2} (resp. n_{-1}) shows the number of times $\widetilde{P^*}(\mathcal{I}_p, \mathcal{I}_{p'}, \Delta, 2)$ returns -2 (resp. -1) when applied in place of $T^*(\Delta)$ in the C^0-test. Column n_{err} shows the number of times the conclusion of $\widetilde{P^*}$ was wrong, and t'_0 shows the total time spent in $\widetilde{P^*}$-tests.

3.2 Using the P^* and $\widetilde{P^*}$-test in a subdivision framework

Our improvement of Algo. 1 is based on the following heuristic remarks. First, it is very unlikely that $\widetilde{P^*}(\mathcal{I}_p, \mathcal{I}_{p'}, \Delta, 2)$ returns -2 (see column n_{-2} in table 2). Second, when $P^*(\mathcal{I}_p, \mathcal{I}_{p'}, \Delta, 2)$ returns $k \geq 0$, it is very likely that Δ contains k roots counted with multiplicity (see column n_{err} in table 2).

We define the C^0-test as follows:

$$C^0(\Delta) := \begin{cases} -1 & \text{if } \widetilde{P^*}(\mathcal{I}_p, \mathcal{I}_{p'}, \Delta, 2) \notin \{-2, 0\}, \\ -1 & \text{if } \widetilde{P^*}(\mathcal{I}_p, \mathcal{I}_{p'}, \Delta, 2) \in \{-2, 0\} \text{ and } T^*(\Delta) \neq 0, \\ 0 & \text{if } \widetilde{P^*}(\mathcal{I}_p, \mathcal{I}_{p'}, \Delta, 2) \in \{-2, 0\} \text{ and } T^*(\Delta) = 0. \end{cases} \tag{8}$$

If $C^0(\Delta)$ is defined in Eq. (8), it returns 0 only if Δ contains no root. Thus Algo. 1 with C^0 and C^*-tests defined by Eqs. (8) and (7) is correct.

Remark now that if a square complex box B of width w does not contain root and is at a distance at least $\frac{3}{2}w$ from a root, then $\Delta(B)$ has isolation ratio

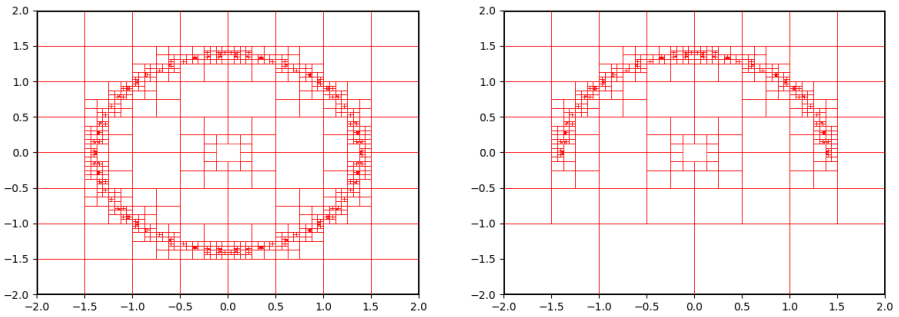

Fig. 2. Computing clusters for $\texttt{Mignotte}_{64}$ in the ROI $[-2,2]+\mathbf{i}[-2,2]$. **Left:** The subdivision tree for Algo. 1. **Right:** The subdivision tree for Algo. 5.

2, and $\widetilde{P^*}(\mathcal{I}_p, \mathcal{I}_{p'}, \Delta(B), 2)$ returns 0 or -2. As a consequence, the termination of Algo. 1 with C^0 and C^*-tests defined in Eqs. (8) and (7) amounts to the termination of Algo. 1 with C^0 and C^* defined in Eq. (4).

4 Clustering roots of polynomials with real coefficients

We consider here the special case where $p \in \mathbb{R}[z]$, and show how to improve a subdivision algorithm for solving the LCP. We propose to leverage on the geometric structure of the roots of p, that are either real, or imaginary and come in complex conjugated pairs: if $\alpha \in \mathbb{C}$ is a root of p so is $\overline{\alpha}$ where $\overline{\alpha}$ is the complex conjugate of α. The modified subdivision algorithm we propose deals only with the boxes of the subdivision tree of the ROI B_0 that have a positive imaginary part; the roots with positive imaginary parts are in the latter boxes. The roots with negative imaginary parts are implicitly represented by the former ones. In Fig. 2 are shown two subdivision trees constructed for clustering roots of a Mignotte polynomial of degree 64; the left-most one is obtained when applying Algo. 1; the right-most one results of our improvement.

Below, we suppose that B_0 is symmetric with respect to the real axis and that p has no root in $2B_0 \setminus B_0$. These two limitations can easily be removed.

Notations Let B be a box centered in c. We define its conjugate $\overline{B}$ as the box centered in $\overline{c}$ with width $w(B)$. We say that B is *imaginary positive* (resp. *imaginary negative*) if $\forall b \in B$, $Im(b) > 0$ (resp. $Im(b) < 0$).

Let $\mathcal{C}$ be a component of boxes of the subdivision tree of B_0. We define $\overline{\mathcal{C}}$ as the component which boxes are the conjugate of the boxes of $\mathcal{C}$. We call *conjugate closure* of $\mathcal{C}$, and we denote it by $\mathcal{C}_{\cup}$ the set of boxes $\mathcal{C} \cup (\overline{\mathcal{C}} \setminus \mathcal{C})$. If $\mathcal{C}$ intersects $\mathbb{R}$, $\mathcal{C}_{\cup}$ is a component. We say that $\mathcal{C}$ is *imaginary positive* (resp. *imaginary negative*) if each box in $\mathcal{C}$ is imaginary positive (resp. imaginary negative).

Algorithm 4 *Quadrisect*($\mathcal{C}$)

Input: A polynomial $p \in \mathbb{R}[z]$ and a component $\mathcal{C}$
Output: A list R of disjoint and not imaginary negative components

$S \leftarrow$ empty list of boxes
for each constituent box B of $\mathcal{C}$ **do**
  for each child B' of B **do**
    if B is **not** imaginary negative **then**
      if $C^0(\Delta(B'), p)$ returns -1 **then**
        $S.push(B')$
$R \leftarrow$ group boxes of S in components
return R

Solving the LCP for polynomials with real coefficients We describe in Algo. 4 a procedure to bisect a component, that discards boxes that are imaginary negative in addition to those that contain no root.

Our algorithm for solving the LCP for polynomials with real coefficients is presented in Algo. 5. It maintains in the queue Q only components of boxes that are imaginary positive or that intersect the real line. Components with only imaginary negative boxes are implicitly represented by the imaginary positive ones. Components that intersect the real line are replaced by their conjugate closure. Components in Q are ordered by decreasing width of their containing boxes. The termination of Algo. 5 is a consequence of the termination of Algo. 1 that is proved in [BSS+16].

Let $\{(\mathcal{C}^1, m^1), \ldots, (\mathcal{C}, m^\ell)\}$ be the list returned by Algo. 5 called for arguments p, B_0, ϵ. Then $\{(\Delta(\mathcal{C}^1), m^1), \ldots, (\Delta(\mathcal{C}^\ell), m^\ell)\}$ is a solution of the LCP problem for p, B_0, ϵ, *i.e.*:

(i) the $\Delta(\mathcal{C}^i)$'s are pairwise disjoint with radius less that ϵ,
(ii) $\forall 1 \leq i \leq \ell$, $(\mathcal{C}^i, m^i)$ satisfies $\#(\Delta(\mathcal{C}^i)) = \#(3\Delta(\mathcal{C}^i)) = m^i$,
(iii) $\texttt{Zero}(B_0, p) \subseteq \bigcup_{i=1}^{\ell} \texttt{Zero}(\Delta(\mathcal{C}^i), p) \subseteq \texttt{Zero}(2B_0, p)$.

In what follow we may write R for the list of connected components in R. $(i), (ii)$ and (iii) are direct consequences of the following proposition:

Proposition 5 *Consider Q and R after any execution of the* **while** *loop in Algo. 5. Decompose Q in two lists Q^1 and Q^2 containing respectively the imaginary positive components of Q and the non imaginary components of Q. Note $\overline{Q^1}$ the list of the conjugates of the components in Q^1 and $Q^2_\cup$ the list of the conjugate closures of the components in Q^2, and let $Q_\cup$ be $\overline{Q^1} \cup Q^2_\cup$. One has:*

(1) any $\alpha \in \texttt{Zero}(B_0)$ is in $R \cup Q \cup Q_\cup$,
(2) any $\mathcal{C} \in R$ is separated from $(R \setminus \{\mathcal{C}\}) \cup Q \cup Q_\cup$,
(3) any $(\mathcal{C}, m)$ in R is such that $m = \#(\Delta(\mathcal{C})) = \#(3\Delta(\mathcal{C}))$.

Proposition 5 is a consequence of Rem. 1 and the following remark.

Remark 6 *Let $p \in \mathbb{R}[z]$ and $\mathcal{C}$ be a component. If $\mathcal{C}$ is imaginary negative or imaginary positive and if there exists m such that $m = \#(\Delta(\mathcal{C})) = \#(3\Delta(\mathcal{C}))$, then $m = \#(\Delta(\overline{\mathcal{C}})) = \#(3\Delta(\overline{\mathcal{C}}))$.*

Algorithm 5 Local root clustering for polynomials with real coefficients

Input: A polynomial $p \in \mathbb{R}[z]$, a ROI B_0, $\epsilon > 0$; assume p has no roots in $2B_0 \setminus B_0$, and B_0 is symmetric with respect to the real axis.
Output: A set R of components solving the LCP.

$R \leftarrow \emptyset$, $Q \leftarrow \{\{B_0\}\}$ *// Initialization*
while Q is not empty **do** *// Main loop*
  $\mathcal{C} \leftarrow Q.pop()$ *//$\mathcal{C}$ has the widest containing box in Q*
  $sFlag \leftarrow$ **false**
  if $\mathcal{C}$ is **not** imaginary positive **then** *//Note: $\mathcal{C} \cap \mathbb{R} \neq \emptyset$*
    $\mathcal{C} \leftarrow \mathcal{C}_{\cup}$
    $sFlag \leftarrow \mathcal{C}$ is separated from Q
  else
    $sFlag \leftarrow$ ($\mathcal{C}$ is separated from Q) **and** $(4\Delta(\mathcal{C}) \cap \overline{\mathcal{C}} = \emptyset)$
  if $w(\mathcal{C}) \leq \epsilon$ **and** $\mathcal{C}$ is compact **and** $sFlag$ **then** *// Validation*
    $m \leftarrow C^*(\Delta(\mathcal{C}), p)$
    if $m > 0$ **then**
      $R.push((\mathcal{C}, m))$
      if C is imaginary positive **then**
        $R.push((\overline{\mathcal{C}}, m))$
      break
  $Q.push(Quadrisect(\mathcal{C}))$ *// Bisection*
return R

5 Numerical results

We implemented the two improvements of Secs. 3 and 4 in `Ccluster`. `ClusterO` refers to the original version of `Ccluster`. Both `ClusterR` and `ClusterPs` implement Algo. 5. In `ClusterPs`, C^0 and C^* are defined by Eqs. (8) and (7).

Testing suite. We tested our improvements on Mignotte and Mandelbrot's polynomials and on Bernoulli and Runnel's polynomials: the Bernoulli polynomial of degree d is $\texttt{Bernoulli}_d(z) = \sum_{k=0}^{d} \binom{d}{k} b_{d-k} z^k$ where the b_i's are the Bernoulli numbers. It has about $d/2$ non-zero coefficients and, as far as we know, cannot be evaluated substantially faster than with Horner's scheme. It has real coefficients, and about $2/3$ of its roots are real or imaginary positive (see left part of Fig. 3). Let $r = 2$, $q_0(z) = 1$, $q_1(z) = z$ and $q_{k+1}(z) = q_k(z)^r + z q_{k-1}(z)^{r^2}$. We define the Runnel's polynomial of parameter k as $\texttt{Runnels}_k = q_k$. It has real coefficients, a multiple root (zero), and can be evaluated fast. The 107 distinct roots of $\texttt{Runnels}_8$ are drawn on right part of Fig. 3.

Results. Table. 5 gives details concerning the execution of `ClusterO`, `ClusterR` and `ClusterPs` for polynomials with increasing degrees. We used $\epsilon = 2^{-53}$ and the ROI $B_0 = [-500, 500] + \mathbf{i}[-500, 500]$ that contains all the roots of all the considered polynomials. Column (#Clus,#Sols) shows the number of clusters and the total multiplicity found. Columns (depth, size) show the depth and the size (*i.e.* number of nodes) of the subdivision tree for each version. t_1, t_2 and t_3

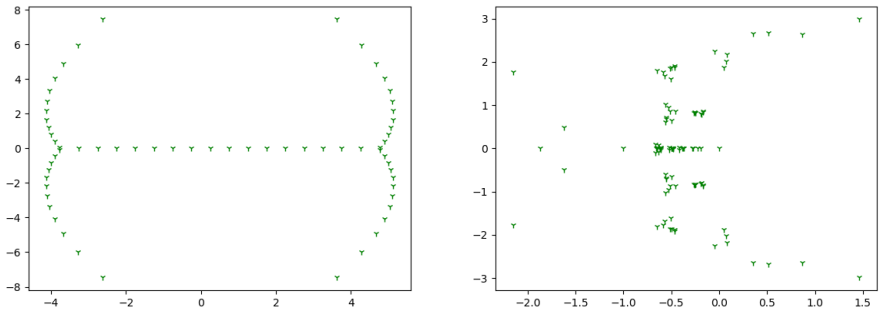

Fig. 3. Left: 64 clusters of roots for the Bernoulli polynomial of degree 64. **Right:** 107 clusters of roots for the Runnel's polynomial of degree 170.

	`CclusterO`			`CclusterR`		`CclusterPs`			
	(#Clus, #Sols)	(depth, size)	t_1	(depth, size)	t_1/t_2	(depth, size)	t_3	t_2/t_3	t_1/t_3
$\texttt{Bernoulli}_{128}$	(128, 128)	(100, 4732)	6.30	(100, 3708)	1.72	(100, 4104)	3.30	1.10	1.90
$\texttt{Bernoulli}_{191}$	(191, 191)	(92, 7220)	20.2	(92, 5636)	1.74	(92, 6236)	10.7	1.08	1.88
$\texttt{Bernoulli}_{256}$	(256, 256)	(93, 9980)	41.8	(93, 7520)	1.67	(91, 8128)	21.9	1.14	1.90
$\texttt{Bernoulli}_{383}$	(383, 383)	(93, 14504)	120	(93, 11136)	1.82	(93, 11764)	53.5	1.23	2.25
$\texttt{Mignotte}_{128}$	(127, 128)	(96, 4508)	5.00	(92, 3212)	1.92	(92, 3484)	1.81	1.43	2.75
$\texttt{Mignotte}_{191}$	(190, 191)	(97, 6260)	15.5	(97, 4296)	2.01	(97, 4688)	4.34	1.77	3.58
$\texttt{Mignotte}_{256}$	(255, 256)	(94, 8452)	31.8	(94, 5484)	2.04	(94, 6648)	10.7	1.44	2.95
$\texttt{Mignotte}_{383}$	(382, 383)	(97, 12564)	79.7	(97, 8352)	1.98	(97, 9100)	26.8	1.49	2.97
$\texttt{Mandelbrot}_{7}$	(127, 127)	(96, 4548)	7.17	(96, 2996)	1.62	(96, 3200)	2.88	1.52	2.48
$\texttt{Mandelbrot}_{8}$	(255, 255)	(96, 8892)	40.6	(96, 5576)	1.71	(96, 6208)	15.1	1.56	2.69
$\texttt{Mandelbrot}_{9}$	(511, 511)	(100, 17956)	266	(100, 11016)	1.89	(100, 11868)	97.1	1.44	2.74
$\texttt{Runnels}_{8}$	(107, 170)	(96, 4652)	13.3	(96, 3252)	1.61	(96, 3624)	6.51	1.26	2.04
$\texttt{Runnels}_{9}$	(214, 341)	(99, 9592)	76.2	(99, 6260)	1.70	(99, 6624)	32.2	1.38	2.36
$\texttt{Runnels}_{10}$	(427, 682)	(100, 19084)	479	(100, 12288)	1.69	(100, 12904)	211	1.34	2.26

Table 3. Details on runs of `CclusterO`, `CclusterR` and `CclusterPs` for polynomials in $\mathbb{R}[z]$ with increasing degree.

stand respectively for the running time in second of `CclusterO`, `CclusterR` and `CclusterPs`.

Algo. 5 achieves speed up t_1/t_2. It is almost 2 for Mignotte polynomials, since about half of its roots are above the real axis. This speed up is less important for the three other families of polynomials, which have a non-negligible ratio of real roots. The speed up achieved by using the P^*-test is t_2/t_3. It is significant for Mignotte's polynomial, which is sparse, and Mandelbrot and Runnel's polynomials for which one can construct fast evaluation procedures.

6 Future works

Our main contribution is a significant practical progress in subdivision root-finding based on a new test for counting roots in a well-isolated disc. If the latter assumption does not hold, the test result is not guaranteed but is very likely to be correct. In a subdivision framework, we have proposed to use a test based on Pellet's theorem to verify its result. We aim to do so by using only

evaluations of p and p'. This would imply a very significant improvement of the root clustering algorithm when p and p' can be evaluated very efficiently.

References

BF00. Dario A Bini and Giuseppe Fiorentino. Design, analysis, and implementation of a multiprecision polynomial rootfinder. *Num. Alg.*, 23(2):127–173, 2000.

BR14. Dario A Bini and Leonardo Robol. Solving secular and polynomial equations: A multiprecision algorithm. *Journal of Computational and Applied Mathematics*, 272:276–292, 2014.

BSS+16. Ruben Becker, Michael Sagraloff, Vikram Sharma, Juan Xu, and Chee Yap. Complexity analysis of root clustering for a complex polynomial. In *Proceedings of the ACM on International Symposium on Symbolic and Algebraic Computation*, ISSAC '16, pages 71–78, New York, NY, USA, 2016. ACM.

BSSY18. Ruben Becker, Michael Sagraloff, Vikram Sharma, and Chee Yap. A near-optimal subdivision algorithm for complex root isolation based on Pellet test and Newton iteration. *J. of Symb. Comp.*, 86:51–96, May-June 2018.

HG69. Peter Henrici and Irene Gargantini. Uniformly convergent algorithms for the simultaneous approximation of all zeros of a polynomial. In *Constructive Aspects of the Fundamental Theorem of Algebra*, pages 77–113. Wiley-Interscience New York, 1969.

IPY18. Rémi Imbach, Victor Y. Pan, and Chee Yap. Implementation of a near-optimal complex root clustering algorithm. In *Mathematical Software – ICMS 2018*, pages 235–244, 2018.

KRS16. Alexander Kobel, Fabrice Rouillier, and Michael Sagraloff. Computing real roots of real polynomials ... and now for real! In *Proceedings of the ACM on International Symposium on Symbolic and Algebraic Computation*, ISSAC '16, pages 303–310, New York, NY, USA, 2016. ACM.

Pan00. Victor Y Pan. Approximating complex polynomial zeros: modified weyl's quadtree construction and improved newton's iteration. *J. of Complexity*, 16(1):213–264, 2000.

Pan02. Victor Y Pan. Univariate polynomials: nearly optimal algorithms for numerical factorization and root-finding. *J. of Symb. Comp.*, 33(5):701–733, 2002.

Pan18. Victor Y Pan. Old and new nearly optimal polynomial root-finders. *arXiv preprint arXiv:1805.12042*, 2018.

PT16. Victor Y Pan and Elias P Tsigaridas. Nearly optimal refinement of real roots of a univariate polynomial. *J. of Symb. Comp.*, 74:181–204, 2016.

Ren87. James Renegar. On the worst-case arithmetic complexity of approximating zeros of polynomials. *J. of Complexity*, 3(2):90–113, 1987.

Sch82. Arnold Schönhage. The fundamental theorem of algebra in terms of computational complexity. *Manuscript. Univ. of Tübingen, Germany*, 1982.

SM16. Michael Sagraloff and Kurt Mehlhorn. Computing real roots of real polynomials. *J. of Symb. Comp.*, 73:46–86, 2016.

XY19. Juan Xu and Chee Yap. Effective subdivision algorithm for isolating zeros of real systems of equations, with complexity analysis. *arXiv preprint arXiv:1905.03505*, 2019.

ZZ19. Vitaly Zaderman and Liang Zhao. Counting roots of a polynomial in a convex compact region by means of winding number calculation via sampling. *arXiv preprint arXiv:1906.10805*, 2019.