How to split a tera-polynomial

François Vigneron, Nicolae Mihalache

How to split a tera-polynomial

Nicolae Mihalache[1] and François Vigneron[2]

February 8, 2024

Abstract

This article presents a new algorithm to compute all the roots of two families of polynomials that are of interest for the Mandelbrot set $\mathcal{M}$: the roots of those polynomials are respectively the parameters $c \in \mathcal{M}$ associated with periodic critical dynamics for $f_c(z) = z^2 + c$ (hyperbolic centers) or with pre-periodic dynamics (Misiurewicz-Thurston parameters). The algorithm is based on the computation of discrete level lines that provide excellent starting points for the Newton method. In practice, we observe that these polynomials can be split in linear time of the degree.

This article is paired with a code library `[Mandel]` that implements this algorithm. Using this library and about 723 000 core-hours on the HPC center *Roméo* (Reims), we have successfully found all hyperbolic centers of period ≤ 41 and all Misiurewicz-Thurston parameters whose period and pre-period sum to ≤ 35. Concretely, this task involves splitting a tera-polynomial, *i.e.* a polynomial of degree $\sim 10^{12}$, which is orders of magnitude ahead of the previous state of the art. It also involves dealing with the certifiability of our numerical results, which is an issue that we address in detail, both mathematically and along the production chain. The certified database is available to the scientific community `[Mand.DB]`.

For the smaller periods that can be represented using only hardware arithmetic (floating points `FP80`), the implementation of our algorithm can split the corresponding polynomials of degree $\sim 10^9$ in less than one day-core. We complement these benchmarks with a statistical analysis of the separation of the roots, which confirms that no other polynomial in these families can be split without using higher precision arithmetic.

Contents

1 Introduction

Algorithms to find all the roots of a given polynomial date back to the beginning of mathematics. Greeks and Babylonians understood quadratic equations 2000 BC and applied them to compute the boundaries of their agricultural fields, to define just taxes and fair trade. Al-khawarizmi's algorithm (825AD) constitutes the official starting point of *algebra* and gave it its name.

At a theoretical level, the fundamental theorem of algebra ensures that every polynomial can be split over $\mathbb{C}$ and the proof of this results was successively refined by d'Alembert (1746), C.F. Gauss (1799), J.R. Argand (1806) and A.L. Cauchy (1821). The race towards general solutions by radicals, though punctuated by great achievements, ultimately stumped all generations until N. Abel (1829) and E. Galois (1832) proved the problem to be unsolvable and discovered a new branch of mathematics (groups theory) along the way.

Polynomial roots are also of key importance for modern numerical analysis. For example, Gauss quadrature formulas provide an extremely efficient way to compute the value of integrals as a linear combination of evaluations at the roots of some orthogonal polynomial (Legendre, Jacobi, Chebyshev, Laguerre, Hermite,...); see *e.g.* [BM92]. Another striking example is the computation of eigenvalues of linear operators, which are, by definition (at least when the ambiant dimension is finite), the roots of the characteristic polynomial. The knowledge of eigenvalues is crucial to understand the behavior of dynamical systems as illustrated, dramatically, by the collapse of Tacoma's bridge in 1940. Similarly, the Covid-19 pandemic has put R_0 (the largest eigenvalue of the regeneration matrix in a population model) under the international spotlight. Eigenvalues are also essential in exploratory data analysis when a predictive model is based on a principal component analysis.

In this article, we focus on two important families of polynomials. The family of *hyperbolic polynomials* is defined recursively by

$$p_0(z) = 0, \qquad p_{n+1}(z) = p_n(z)^2 + z. \tag{1}$$

For $n \geq 1$, one has $\deg p_n = 2^{n-1}$. The family of *Misiurewicz-Thurston polynomials* is a two parameters family defined for $\ell, n \in \mathbb{N}$ by

$$q_{\ell,n}(z) = p_{\ell+n}(z) - p_\ell(z). \tag{2}$$

The polynomials p_n and $q_{\ell,n}$ are closely related to the Mandelbrot set $\mathcal{M}$ (see [CG93], [Mil90] or Section 2 for further details). This set is both one of the most intricate mathematical objects and a very famous computer-generated image (Figure 1). Its fractal nature has fascinated the general public, the mathematical and the computer science communities since 1980 and a great deal of research has been done around its properties during this time. The century old conjecture of P. Fatou [Fat19] remains largely open and can be stated as the following question: are all connected components of the interior of $\mathcal{M}$ hyperbolic, *i.e.* do they each contain exactly one root of some p_n ?

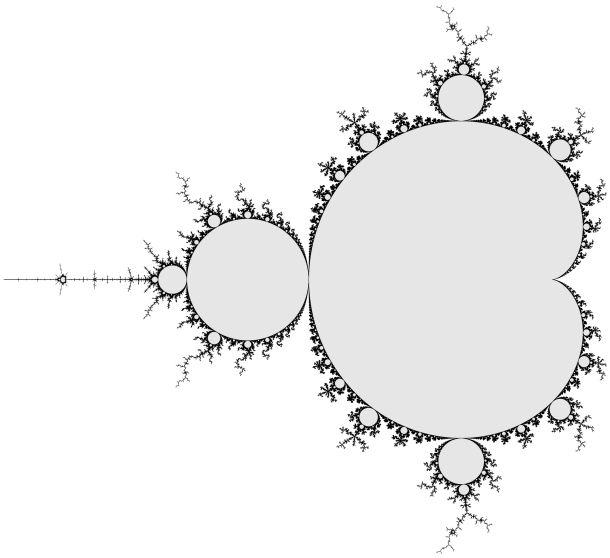


Figure 1: *The Mandelbrot set $\mathcal{M}$ with $\partial\mathcal{M}$ in black and in gray, the interior of $\mathcal{M}$.*

The current state of the art for root-finding algorithms is limited, in practice, to polynomials whose degree does not exceed a few millions or, at best, a few hundred millions. In Section §3 we survey briefly the corresponding algorithms. For example, D. Schleicher and R. Stoll [SS17] have mastered this scale and give computing times[1] ranging from 18 hours-core to 15 days-core for various polynomials of degree $2^{20} \sim 10^6$. Other implementations using a different splitting algorithm and massively parallel architectures [GSCG17] end up with similar core requirements and degree limitations. Eigenvalue methods encounter similar limitations on the size of non-zero elements in the matrices [DT93], [IYM11], [SZI+17], [SK19]. A substantial leap forward was accomplished by [RSS17]: using an adaptative mesh-refinement technique for the Newton method they split a polynomial of degree 2^{24} in less than 7 days-core, however with a few missing roots, and even one instance of a polynomial of degree $2^{30} \sim 10^9$ in 4 days-core. This remarquable implementation is even backed by theoretical results [HSS01], [Sch23] (see §3.5 below) that ensure that a certain set of starting points guaranties that all roots will be found.

For the application to Gaussian quadrature, one can exploit the specific connection between orthogonal polynomials and the associated ODEs to compute the roots in linear time. The published benchmark time [Bre17] is about 6 day-core to compute a Gauss-Jacobi rule of record size 10^{12}. The algorithm is explained in [BMF12], [HT13], [TTO16].

[1] A *time-core* unit is a rough measure of the practical complexity of an implementation, regardless of the architecture of the computer. It is obtained by adding the individual compute time of each active computing unit involved throughout the computation. A convenient estimate and upper bound is the product of the wall-clock time of the computation by the number of CPU cores in use. Up to a point, mostly limited by memory management, parallelization leads to a better wall-clock time, even though the overall time-core requirements may be substantially higher.

Let us point out that, while numerical analysis has recently reached problems with 2×10^9 degrees of freedom for hyperbolic PDEs in field-ready medical imaging [TAB+19], 550×10^9 degrees of freedom for elliptic PDEs on the *Titan* supercomputer [GMSB16] and even 10^{15} degrees of freedom for a turbulence simulation on *Summit* [RAY19], those remarkable achievements do not yet imply the ability to solve routinely eigenvalue and polynomial splitting problems of this size. In particular, as we will see in Section §6, the low separation between roots of polynomials of extreme degree cannot, in general, be resolved using hardware floating point arithmetic.

1.1 Our results in a nutshell

This article raises the state of the art of polynomial splitting to a degree about one trillion (*i.e.* $2^{40}\sim10^{12}$), called a *tera-polynomial*[2]. We do not claim full generality and focus on the families p_n and $q_{\ell,n}$ defined above.

To achieve this result without brute force, the main idea is a new splitting algorithm. Roughly speaking, the classical Newton's method consists in choosing [HSS01] a mesh of initial points that encircles the roots and to iterate the Newton map until the trajectories reach the roots; this method results in a dense mesh of most of the disk (see Figure 2).

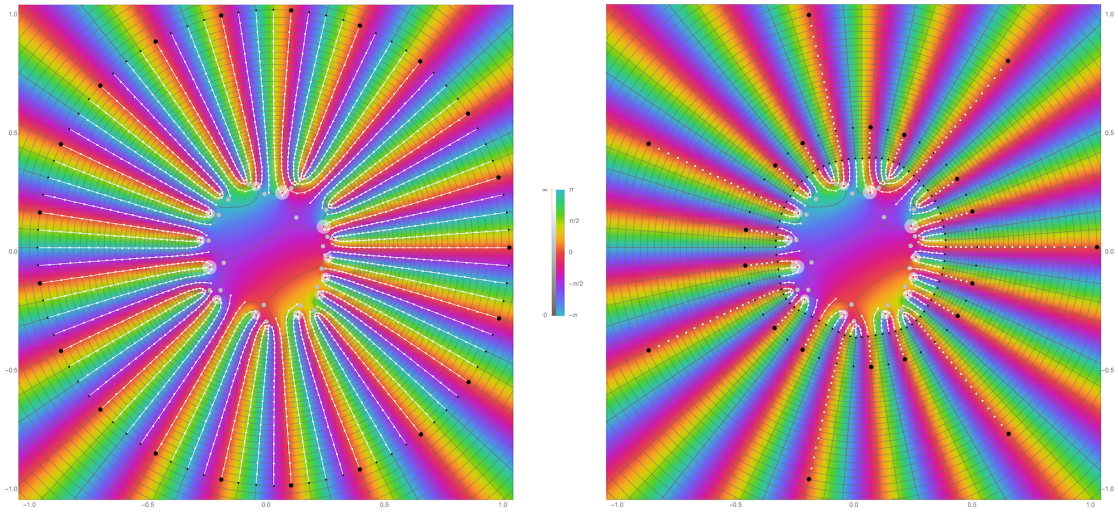


Figure 2: *Newton's method to split a polynomial of degree $d=20$ with 4 starting points (black) per root. The trajectories (white) are Newton's iterations, which are stoped in case of divergence; the gray dots mark the critical points; the white disk are contracted to the roots (red). The classical choice for the starting points (left) leads to $O(d^2)$ Newton steps. A better choice, proposed in this article (right) optimizes the position of the starting points by combining their computation with that of a level curve and brings, in practice, the total number of steps to $O(d)$. See also [RSS17, Fig. 2] for a similar idea, though implemented differently.*

[2] As a trinomial denotes a polynomial expression with 3 terms, we should call it a *teranomial*. We decided against it as the single letter difference with the prefix *tetra* (that denotes 4) could lead to confusion.

Instead, we propose to choose the starting points of the Newton method on a level curve that hugs the set of roots tightly. To produce this curve, one starts on an extremely lean mesh to perform most of the "descend" with as little computations as possible. When needed, one can then densify the mesh at a certain level set (horizontally) and then continue the descend (vertically) with more points. Ultimately, we catch up with the dense mesh of the classical method, having only a few Newton steps left until we find all the roots.

We are releasing an exhaustive and proven list [Mand.DB] of all the roots of p_n (called hyperbolic centers) for n up to period 41 and all the roots of $q_{\ell,n}$ (called pre-periodic or Misiurewicz-Thurston points) of order $\ell + n \leq 35$. By proven, we mean that each point z_j in our database comes with a mathematical statement that guaranties two things: an exact root $\tilde{z}_j$ lies within a certain maximal tolerance from our numerical value, and a standard refinement technique (Newton's method) applied to z_j converges to $\tilde{z}_j$. To ensure a strict control of the computational errors that is robust enough to withstand intensive iterations, we have revisited and extended the theory of disk arithmetic.

As a companion to this article, we also release an implementation of our algorithm as a standard C library [Mandel]. This library can be used to check and explore our database [Mand.DB] and contains all the tools necessary to perform extractions, generate images, compute arbitrary precision refinements, etc. This library is optimized on all fronts: it can run on a single core for the smallest periods or can take advantage of the massively parallel architectures of modern HPC centers for the higher ones. For example, the splitting of p_{21}, which is reported as requiring at least 18.8 hours-core in [SS17] takes only 10 seconds with our algorithm, on a single core of a consumer-grade personal computer. Similarly, we can split p_{29} in only 1 hour-core and p_{33} in about 1 day-single-core. The computations automatically switch to high-precision, using the [MPFR] library, but only when necessary. For example, when $n \geq 30$ the roots of p_n near -2 are so close that they cannot be resolved in the standard 64 bits floating point arithmetic. Up to $n \leq 41$, we use numbers with 128 significant bits. Throughout the library, custom data structures ensure near optimal memory usage, data access and computation times.

The case of the polynomials $q_{\ell,n}$ is interesting because, contrary to the p_n, they have multiple roots, which slow down drastically the dynamics of the Newton map. However, our algorithm remains robust in this case. We stopped at the order $\ell + n \leq 35$. The volume of data generated by the roots of all (ℓ, n) combinations is of the same order of magnitude to that of the roots of all p_n for $n \leq 39$ (see Section 2.4).

Our database and its companion library constitute a powerful *numerical microscope* whose resolution extends far beyond the current state of the art and that can assist the mathematical community in the exploration of the properties of $\mathcal{M}$. While an in-depth analysis will take time, we already present in this first article a few properties, like the minimal distance between the roots of a given type (see Section 6.3.2).

1.2 Structure of this article

Our article is organized as follows. Section §2 contains a brief self-contained introduction to the Mandelbrot set. Section §3 presents a quick review of the most common splitting (or at least root finding) algorithms for polynomials. A detailed presentation of our new algorithm is done in Section §4, along with a general discussion of its algorithmic and bit complexity. As we do not claim full generality, the specifics of working with polynomials tailored to the Mandelbrot set is discussed too. Readers interested in polynomials of high degrees may appreciate, on the side, our improvement of the standard evaluation scheme [AMV22], along with its companion library [FPE].

Section §5 is focused on the question of providing certifiable results. The backbone of this section is a theory of a disk arithmetic, which, contrary to interval arithmetic, tolerates a high number of iterations.

Our implementation [Mandel] is presented in Section §6. We discuss the practical cost of splitting the tera-polynomial p_{41} and confirm, in practice, the complexity claims of Section §4. A listing of the tasks is given in Appendix D.

A proof of our Theorem 2 stated in Section §2 on the factorization of $q_{\ell,n}$ is given in Appendix A. A few background auxiliary mathematical results used in section §5 are recalled very briefly in Appendix B. For the sake of convenience, the notations introduced throughout the article are listed in Appendix C.

1.3 Thanks

We are grateful to *Romeo*, the HPC center of the University of Reims Champagne-Ardenne, for hosting our project graciously even though its scale (near 50TB of data and 1 century-core / 870 kh of CPU usage for both the development and production phase) is quite substantial for this regional structure.

It is our pleasure to thank our families who supported us in the long (5+ years) and, at times stressful, process that gave birth to the `Mandelbrot` library. We also thank Sarah Novak for a thorough proof-reading of this article.

2 About the Mandelbrot set

In this section, we will collect the strict minimum on the Mandelbrot set and fix notations that will be needed subsequently. See Appendix C for standard notations.

The *Mandelbrot* set $\mathcal{M}$ is composed of the parameters $c \in \mathbb{C}$ for which the sequence $(p_n(c))_{n\in\mathbb{N}}$ remains bounded (Figure 1). This sequence is the orbit by iterated compositions of the only critical point $z = 0$ of the map $f_c(z) = z^2 + c$, *i.e.*

$$\forall n \in \mathbb{N}, \qquad f_c^n(0) = \underbrace{f_c \circ \ldots \circ f_c}_{n \text{ times}}(0) = p_n(c). \tag{3}$$

For short, the sequence $(p_n(c))_{n\in\mathbb{N}}$ is called the *critical orbit* of f_c. $\mathcal{M}$ is a connected set, whose complement in $\overline{\mathbb{C}}$ is simply connected [DH82], [Sib84]. One has

$$\mathcal{M} \subset \mathrm{D}(0,2) \cup \{-2\}.$$

There are many tools online to explore $\mathcal{M}$, both for scientific purposes or just for its intrinsic beauty; see *e.g.* [Che10] or [Guc].

For $c \in \mathbb{C}$, the dynamics of f_c splits $\overline{\mathbb{C}}$ in two complementary sets. The *Fatou* set $\mathcal{F}_c$ is the open subset of $z \in \overline{\mathbb{C}}$ in the neighborhood of which, the sequence $(f_c^n)_{n\in\mathbb{N}}$ is a normal family *i.e.* precompact in the topology of local uniform convergence. On the contrary, on the *Julia* set $\mathcal{J}_c = \overline{\mathbb{C}} \backslash \mathcal{F}_c$, the dynamics appears to be, loosely speaking, chaotic. Both $\mathcal{F}_c$ and $\mathcal{J}_c$ are fully invariant (*i.e.* invariant sets of the forward and backwards dynamics) and that $c \in \mathcal{M}$ if and only if $\mathcal{J}_c$ is connected. For a review of the properties of Fatou and Julia sets, see *e.g.* [CG93], [Mil90] for polynomial and rational maps and [Ber93], [MPRW22] for entire and meromorphic functions.

2.1 Hyperbolic parameters

For $n \geq 1$, the roots of p_n are parameters $c \in \mathcal{M}$, called *hyperbolic centers*, whose critical orbits are periodic of period n. The roots of p_n are simple [DH82, Buf18] and the polynomial $p_n(z)$ is divisible by $p_k(z)$ for any divisor k of n. A. Douady and J.H. Hubbard [DH82, DH85] have shown that the hyperbolic centers are interior points of $\mathcal{M}$, with at most one hyperbolic center per connected component of the interior. The set of all hyperbolic centers is also dense in the boundary of $\mathcal{M}$ in the sense that its closure in $\mathbb{C}$ contains $\partial\mathcal{M}$.

Let us define the set of hyperbolic centers of *order* $n \geq 1$ as the subset of $p_n^{-1}(0)$ whose minimal (or fundamental) period is exactly n; in other terms:

$$\mathrm{Hyp}(n) = \left\{ z \in p_n^{-1}(0) \,\middle|\, \forall k \in \mathrm{Div}(n)^*,\ p_k(z) \neq 0 \right\} \tag{4}$$

where $\mathrm{Div}(n) = \{k \in \mathbb{N}^* \,;\, \exists k' \in \mathbb{N},\quad kk' = n\}$ and $\mathrm{Div}(n)^* = \mathrm{Div}(n)\backslash\{n\}$ is the set of strict divisors of n. For example, $\mathrm{Hyp}(1) = \{0\}$ and $\mathrm{Hyp}(2) = \{-1\}$. The reduced hyperbolic polynomial, also known as Gleason's polynomial, is:

$$h_n(z) = \prod_{r \in \mathrm{Hyp}(n)} (z - r). \tag{5}$$

The polynomials p_n and h_n have integer coefficients. While expected, the irreducibility of h_n over $\mathbb{Z}[z]$ remains conjectural (see [HT15, last remark of §3], [SS17, p.155]).

Theorem 1 (folklore, included in [HT15])**.** *The complete factorization of* p_n *is*

$$p_n(z) = \prod_{k|n} h_k(z). \tag{6}$$

Moreover, the cardinal of $\mathrm{Hyp}(n)$ *is given by*

$$|\mathrm{Hyp}(n)| = \sum_{k|n} \mu(n/k) 2^{k-1} \tag{7}$$

where μ *is the Möbius function, i.e.*

$$\mu(n) = \begin{cases} (-1)^\nu & \text{if } n \text{ is square free and has } \nu \text{ distinct prime factors,} \\ 0 & \text{if } n \text{ is not square free.} \end{cases}$$

The notation $k|n$ means $k \in \mathrm{Div}(n)$ and $|S|$ denotes the cardinal of a finite set. The identity (6) is well known and is key in the count of hyperbolic centers. The Online Encyclopedia of Integer Sequences [OEIS] attributes (7) to Warren D. Smith and Robert Munafo (2000), sadly with no published reference. For a more general result that applies to iterated maps of the form $z^d + c$, see [HT15].

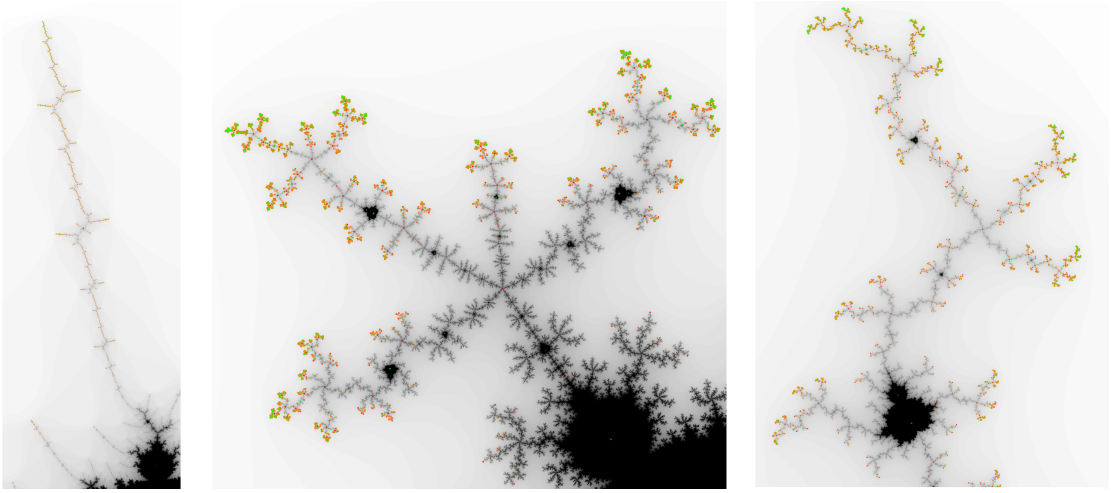


Figure 3: The hyperbolic points $\mathrm{Hyp}(n)$ for $n \le 18$ in green and the Misiurewicz-Thurston parameters $M_{\ell,n} = \mathrm{Mis}(\ell, n)$ with $\ell + n \le 16$ in red in different parts of the Mandelbrot set $\mathcal{M}$, with $\mathrm{Re}\, c$ increasing from left to right between images.

2.2 Pre-periodic or Misiurewicz-Thurston parameters

For integers $n \ge 1$ and $\ell \ge 2$, the roots of $q_{\ell,n}$ are parameters $c \in \partial\mathcal{M}$, called *pre-periodic centers* or *Misiurewicz-Thurston* points, whose critical orbits are pre-periodic, that is it becomes periodic of period n after the first ℓ steps. For $\ell \in \{0, 1\}$, one can check immediately that

$$q_{0,n}(z) = p_n(z) \qquad \text{and} \qquad q_{1,n}(z) = p_n^2(z). \tag{8}$$

The dynamical reason for the second identity in (8) is that 0 is the only pre-image of c under f_c so a pre-periodic orbit of type $(1, n)$ starting at zero actually loops back to zero,

so is periodic of period n. The polynomial $q_{\ell,n}(z)$ is divisible by $q_{\ell,k}(z)$ for any divisor k of n and by $q_{\ell',n}(z)$ for any $\ell' \leq \ell$. Some of the roots of $q_{\ell,n}(z)$ have multiplicity; however, the polynomial

$$s_{\ell,n}(z) = \frac{q_{\ell,n}(z)}{q_{\ell-1,n}(z)} \in \mathbb{Z}[z] \tag{9}$$

has simple roots [Buf18], [HT15, Lemma 3.1]. Douady-Hubbard [DH82, DH85] have shown that the set of all pre-periodic points is dense in the boundary of $\mathcal{M}$. Visually, those points are either branch tips, centers of spirals or points where branches meet (see Figure 3).

By analogy with the hyperbolic case, let us define the set of *Misiurewicz points of type* (ℓ, n) as the subset of $q_{\ell,n}^{-1}(0)$ whose dynamical parameters are exactly ℓ and n; in other terms:

$$\mathrm{Mis}(\ell, n) = \left\{ z \in q_{\ell,n}^{-1}(0) \;\middle|\; \begin{array}{l} q_{\ell-1,n}(z) \neq 0, \\ \forall k \in \mathrm{Div}(n)^*,\ q_{\ell,k}(z) \neq 0 \end{array} \right\}. \tag{10}$$

We call $\ell + n$ the *order* of $\mathrm{Mis}(\ell, n)$ because $q_{\ell,n}$ is a polynomial of degree $2^{\ell+n-1}$. The reduced polynomial whose roots are exactly $\mathrm{Mis}(\ell, n)$ is denoted by

$$m_{\ell,n}(z) = \prod_{r \in \mathrm{Mis}(\ell,n)} (z - r) \in \mathbb{Z}[z]. \tag{11}$$

Note that $\mathrm{Mis}(0, n) = \mathrm{Hyp}(n)$ and $\mathrm{Mis}(1, n) = \emptyset$ because of (8). The count of the Misiurewicz points has been established by B. Hutz and A. Towsley [HT15, Cor. 3.3]:

$$|\mathrm{Mis}(\ell, n)| = \Phi(\ell, n)\, |\mathrm{Hyp}(n)| \tag{12}$$

where

$$\Phi(\ell, n) = \begin{cases} 1 & \text{if } \ell = 0, \\ 2^{\ell-1} - 1 & \text{if } \ell \neq 0 \text{ and } n | \ell - 1, \\ 2^{\ell-1} & \text{otherwise.} \end{cases}$$

To get a complete factorization of $q_{\ell,n}(z)$ in terms of (5) and (11), one needs to understand the multiplicity of the hyperbolic factors. We claim the following result.

Theorem 2. *For $\ell \in \mathbb{N}$ and $n \in \mathbb{N}^*$, one has*

$$q_{\ell,n}(z) = \prod_{k|n} \left(h_k(z)^{\eta_\ell(k)} \prod_{j=2}^{\ell} m_{j,k}(z) \right) \tag{13}$$

where the multiplicity $\eta_\ell(k)$ is given by[3]

$$\eta_\ell(k) = \left\lfloor \frac{\ell - 1}{k} \right\rfloor + 2. \tag{14}$$

[3] In (14), the floor function is denoted by $\lfloor x \rfloor = k$ if $k \in \mathbb{Z}$ and $x \in [k, k+1)$.

In other words, the roots of $q_{\ell,n}$ are composed of the points $\mathrm{Hyp}(k)$ *for any divisor k of n, which are roots with multiplicity $\eta_\ell(k)$, and of the points* $\mathrm{Mis}(j,k)$ *for $2 \le j \le \ell$, which are simple roots.*

This result appears to be new. We propose a direct proof in Appendix A.

2.3 Harmonic measure

The sets of all hyperbolic-centers and of all Misiurewicz-Thurston points are respectively denoted by:

$$\mathrm{Hyp} = \bigcup_{n\in\mathbb{N}^*} \mathrm{Hyp}(n) \subset \overset{\circ}{\mathcal{M}} \qquad \text{and} \qquad \mathrm{Mis} = \bigcup_{\substack{\ell\ge 2\\ n\in\mathbb{N}^*}} \mathrm{Mis}(\ell,n) \subset \partial\mathcal{M}.$$

As mentioned above, $\overline{\mathrm{Mis}} = \partial\mathcal{M} \subset \overline{\mathrm{Hyp}}$. The density measures of Hyp and Mis (*i.e.* the limit of the uniform counting measure on finite subsets of increasing order) both coincide with the *harmonic measure* for $\partial(\mathcal{M}^c)$, which is the image of the uniform measure on the unit circle under the Riemann map $D(0,1)^c \to \mathcal{M}^c$. Intuitively, the harmonic measure expresses the asymptotic density of the external rays as they land on the boundary of the Mandelbrot set. The support of the harmonic measure is the only subset of $\partial\mathcal{M}$ that is "visible" to a random brownian motion starting at ∞ and is of Hausdorff dimension 1. For further details, see [Mak85], [Lev90], [BS05], [FG15], [GV17], [GV19] and Remark 7 below.

2.4 Scale of the computational challenge

Using the library `[Mandel]` associated with this article, we have completely determined the sets $\mathrm{Hyp}(n)$ for all $n \le 41$ and $\mathrm{Mis}(\ell,n)$ for $\ell + n \le 35$. Both problems consists in solving polynomial equations, namely (1) or (2), of respective degrees 2^{n-1} or $2^{\ell+n-1}$. Overall, we have sifted through about 3.3×10^{12} polynomial roots and identified 2 725 023 424 662 distinct parameters of the Mandelbrot set. About 80% of them are hyperbolic centers.

The tera-polynomial p_{41} has degree 1 099 511 627 776 and, as 41 is a prime number,

$$p_{41}^{-1}(0) = \mathrm{Hyp}(41) \cup \{0\}.$$

More generally, the deflation of $\mathrm{Hyp}(n)$ due to strict divisors of n constitutes an asymptotically negligible subset of the roots of p_n.

The situation is not exactly the same for $\mathrm{Mis}(\ell,n)$. There are 33 non-hyperbolic preperiodic polynomials $q_{\ell,n}$ of order $\ell + n = 35$, each of degree 17 179 869 184. However, the exact count 271 590 481 057 of $\bigcup \mathrm{Mis}(\ell,n)$ for all types of order 35 represents only 47.9% of the total number of roots of those polynomials. More generally, (12) ensures that, for large N:

$$\sum_{\substack{\ell+n=N\\ \ell\ne 0}} |\,\mathrm{Mis}(\ell,n)| \simeq \frac{N-2}{2}\,|\,\mathrm{Hyp}(N)|.$$

The deflation of $\mathrm{Mis}(\ell, n)$ due to divisors is therefore, asymptotically, 50% as $\ell+n \to \infty$.

To convey a sense of the orders of magnitude involved for storing and processing the roots of polynomials of giga- and tera-scale, let us underline that a computation that takes, on average, only 1 milli-second per root requires about 12 days-core for a polynomial of degree 10^9. This requirement jumps to 32 years-core ($\sim$ 280k hours-core) for a polynomial of degree 10^{12}.

Our actual computation times are discussed in Section §6. The success of our entreprise is due to the efficiency of our new splitting algorithm for p_n and $q_{\ell,n}$ (see Section 4), which is bounded by $O(d \log d)$ and runs as $O(d)$ in practice.

3 A brief review of root finding algorithms

In the rest of the article, one will assume that $P \in \mathbb{C}[z]$ is a polynomial of degree $d = \deg P \geq 1$ and that its roots $\mathcal{Z} = P^{-1}(0)$ are localized in the disk $\mathrm{D}(0, r)$. See Theorem 10 in Appendix B to estimate r from the coefficients of P.

A *splitting algorithm* is an algorithm that provides the list of all the roots of P through convergent numerical approximations. Splitting algorithms can be classified in three broad families: methods based on Newton's iterations, root isolation methods and eigenvalue methods. Let us review them briefly.

We denote by V_d the *arithmetic complexity* of evaluating a polynomial of degree d. Horner's method ensures that, in general, $\mathrm{V}_d = O(d)$ when the coefficients are known. Some evaluation schemes offer a similar complexity with better balanced intermediary computations, like Estrin's method, which is implemented in the Flint library [FLINT], or some variants [Mor13]. Other schemes take advantage of multipoint evaluations (see *e.g.* [KS16]). In a separate article [AMV22], we propose a *Fast Polynomial Evaluator* algorithm and its practical implementation [FPE] that brings the average cost of evaluating real and complex polynomials with a fixed precision down to $\mathrm{V}_d = O(\sqrt{d \log d})$ after a preprocessing phase that can be performed independently of the precision[4] and that costs $O(d \log d)$.

For iteratively defined polynomials (*e.g.* for p_n, which is of degree $d = 2^{n-1}$), one obviously has $\mathrm{V}_d = O(\log_2 d)$. To keep comparisons fair between algorithms, we will therefore explicitly factor out the cost of evaluations whenever possible.

Remark 1. *If a polynomial P has simple roots, the computational cost of checking that a sorted list of numbers is indeed a list of all the roots of P is $O(d\,\mathrm{V}_d)$. To check that a root z is of multiplicity k, one computes $P(z)$, $P'(z)$, $\ldots$, $P^{(k-1)}(z)$ instead of computing k distinct values of P so the contribution to the total cost remains of order $k\,\mathrm{V}_d$ provided that each derivative can also be evaluated in time $O(\mathrm{V}_d)$. The cost of checking roots thus remains $O(d\,\mathrm{V}_d)$.*

[4] In the pre-processing phase of the FPE algorithm, only the integer part of the base-2 logarithm of the coefficients is needed.

3.1 A word on numerical instabilities

A naive splitting algorithm consists in finding a root of P by any available method, and then reduce the degree of P by performing a euclidian division.

In practice, this method is plagued by numerical instabilities, illustrated on Wilkinson's polynomial [Wil84]

$$W_{20}(x) = \prod_{n=1}^{20}(x-n)$$

whose largest root is 20^{19} times less stable than its smallest root when the coefficients are perturbed in the canonical basis (though other basis may behave better). In general, determining the roots of a polynomial from the list of its coefficients is a highly ill-conditioned problem.

Consequently, for very large polynomials, computing the coefficients is not always advisable. For example, as $p_3(z) = z^4 + 2z^3 + z^2 + z$, the largest coefficient of $p_n(z)$ for $n \geq 3$ exceeds $2^{2^{n-3}}$ while the smallest one remains 1. Storing the coefficients of $p_{41}(z)$ exactly is, at best, impractical and any reasonable approximation of those coefficients would not be sufficiently precise to compute the roots accurately.

3.2 Root isolation methods

Bracketing (or root isolation) consists in identifying an interval of the real line or a domain of the complex plane where the number of roots of P is prescribed.

Methods based on this principle are usually developed on the real line where they constitute a textbook application of the intermediary value theorem (bisection method, or the more involved ITP method).

In the complex plane, the splitting circle method and the Lehmer–Schur algorithm are based on the residue theorem, which implies that, for all $n \in \mathbb{N}$

$$\sum_{\substack{z\in\Omega \\ P(z)=0}} m_P(z) z^n = \frac{1}{2i\pi}\int_{\partial\Omega} \frac{P'(z)}{P(z)} z^n dz$$

for any smooth domain Ω whose boundary avoids the zeros of P and where $m_P(z)$ denotes the multiplicity of z as a root of P. Successive refinements of Ω will either isolate single roots or clusters of close roots that can be identified using Newton's identities. Aside from the cost of computing the integral to a sufficient precision, those methods are plagued by the geometrical problem of finding a "good" circle, *i.e.* one which is proper to initiate a sequence of divide and conquer iterations.

Of similar nature is Graeffe's method, where the sequence

$$P_{n+1}(z^2) = (-1)^d P_n(z) P_n(-z)$$

is computed from $P_1 = P$. The roots of P_n are $\{z^{2^n}\,;\, P(z) = 0\}$ and are thus, generically, exponentially separated from each other. In that case Vieta's formulas can easily be inverted approximately, which gives the roots of P_n and ultimately, those of P.

Root isolation methods excel in finding roots of a given polynomial, but they are not a splitting algorithms per se. We will not insist on comparing their complexity.

Our algorithm (see §4 below) presents a strong familiarity with the splitting circle method: the level lines that we will use provide a tight enclosure of the roots of P by a Jordan curve. Tighter level sets would enclose the roots in a finite union of disjointed connected sets (see Figures 2 and 6).

3.3 Eigenvalue algorithms

Splitting a polynomial P can always be restated as the question of finding all the eigenvalues of the companion matrix $A_P \in \mathcal{M}_d(\mathbb{C})$, at least provided that the coefficients of P are known.

$$A_P = \begin{pmatrix} 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \ddots & \vdots \\ \vdots & & \ddots & \ddots & 0 \\ 0 & \cdots & \cdots & 0 & 1 \\ c_0 & c_1 & \cdots & \cdots & c_{n-1} \end{pmatrix} \qquad \text{if} \qquad P(x) = x^n - \sum_{k=0}^{n-1} c_k x^k.$$

The vector $(1, x, x^2, \dots, x^{n-1})$ is an eigenvector of A_P if and only if $P(x) = 0$.

The complexity of eigenvalue algorithms usually depends on the structure of the matrix under consideration. For example, the modern version of the QR algorithm for Hessenberg matrices[5] costs $O(d^2)$ per iteration [BDG04], [CGXZ07]. The sequence starts with $A_0 = A$, the matrix whose spectrum is being computed. The principle of each step consists in an orthogonal similarity transform $A_{k+1} = Q_k^T A_k Q_k$ where

$$A_k = Q_k R_k \quad \text{with} \quad \begin{cases} Q_k \text{ orthogonal} \\ R_k \text{ upper triangular} \end{cases}$$

As $Q_k^T Q_k = I$, it boils down to computing Q_k, R_k and then forming $A_{k+1} = R_k Q_k$. The number of iterations used in practice until A_k reaches the Schur form of A (upper triangular) may be $O(d)$ or higher. Companion matrices belong to the Hessenberg class, so, as a splitting algorithm, the QR method costs at least $O(d^3)$.

Note that the polynomial P is not explicitly evaluated here, so the factor V_d is not included in the arithmetic complexity. However, it appears implicitly in each step as a cost of matrices operations.

[5] A Hessenberg matrix has zero entries above the first superdiagonal, which is the case for A_P.

Remark 2. *Many applications that require polynomial splitting arise naturally as an eigenvalue problem. For example, in structural mechanics, the asymptotic stability analysis of an engineering design calls for the determination of the spectrum of some PDE operator, which can be approximated by a large band-limited finite-elements matrix. Most linear algebra algorithms are tailored for the needs of the specific applications, like finding the extreme eigenvalues or their distribution [ACE09], or determining the eigenvectors. They do not always constitute proper splitting algorithms, even though they can occasionnaly be repurposed as such.*

3.4 Newton's iterations

The core of many iterative splitting algorithms are Newton's iterations:

$$z_{n+1} = N_P(z_n) \qquad \text{where} \qquad N_P(z) = z - \frac{P(z)}{P'(z)}. \tag{15}$$

The fixed points of the dynamics of N_P are the roots of P.

In the vicinity of each simple root, this method converges bi-exponentially. More precisely, if $z_* = \lim z_n$ with $P'(z_*) \neq 0$, Taylor's formula for P can be rewritten

$$z_{n+1} - z_* = \frac{(z_n - z_*)^2}{P'(z_n)} \int_0^1 (1-t) P''(tz_* + (1-t)z_n) dt.$$

For $\rho > 0$, one defines $C_\rho = \frac{1}{2} \sup |P''(z)| / \inf |P'(z)|$, each extremum being computed for $z \in \mathrm{D}(z_*, \rho)$. One can always assume that $\rho C_\rho < 1$ because $C_\rho \to \frac{1}{2} \left| \frac{P''(z_*)}{P'(z_*)} \right| \in \mathbb{R}_+$ as $\rho \to 0$, so $\rho C_\rho \to 0$. In that case, if $z_{n_0} \in \mathrm{D}(z_*, \rho)$ then

$$\forall n \geq n_0, \quad |z_{n+1} - z_*| \leq C_\rho |z_n - z_*|^2 \quad \textit{i.e.} \quad |z_n - z_*| \leq C_\rho^{-1} (\rho C_\rho)^{2^{n-n_0}}. \tag{16}$$

When the root z_* is of multiplicity $\mu \geq 2$, then the convergence of Newton's algorithm is only exponential. Indeed, one has

$$N_P'(z) = \frac{P(z)P''(z)}{P'(z)^2} \underset{z \to z_*}{\longrightarrow} 1 - \frac{1}{\mu} \in \left[\tfrac{1}{2}, 1\right)$$

so for any z_0 in the bassin of attraction

$$\mathcal{A}(z_*) = \{z \in \mathbb{C} \,;\, \lim_{n \to \infty} N_P^n(z) = z_*\}$$

the estimate (16) is replaced, using Taylor's formula for N_P, by

$$\frac{z_{n+1} - z_*}{z_n - z_*} \to 1 - \frac{1}{\mu}. \tag{17}$$

In that case, the accelerated scheme

$$z'_{n+1} = z'_n - \mu \cdot \frac{P(z'_n)}{P'(z'_n)} \tag{18}$$

could be used instead to restore an ultimately bi-exponential behavior in the vicinity of multiple roots. In practice, however, μ may be unknown and, even if it was known, this accelerator will completely destroy the attraction bassins of the simple roots, as illustrated on Figure 4. A better accelerator may be König's method [BH03] that replaces P/P' in (15) by ratios of consecutive higher-order derivatives of $1/P$.

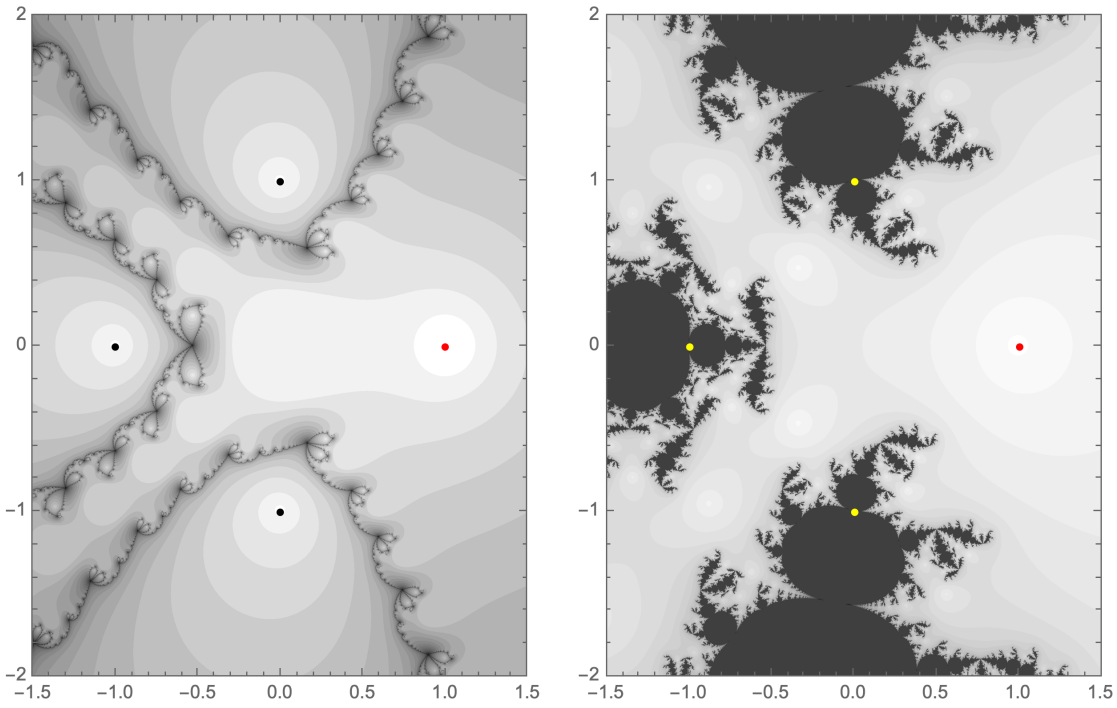


Figure 4: Attraction bassins and Julia set of the Newton method (left) and of the accelerated method with $\mu = 2$ (right) for $P(z) = (z-1)^2(z+1)(z^2+1)$.

The so called, quasi-Newton variants (Muller, Steffensen) avoid computing the derivative for numerical purposes, in particular if the method is applied to a general function whose derivative is not known or not easily computable. A typical example is the secant method. Another class of quasi-Newton methods involve higher-order derivatives (Halley, Householder, Laguerre, Jenkins-Traub,...): they give a higher-order of convergence in exchange for a tighter requirement on the smoothness of the function and higher-order norms in the estimates. See *e.g.* [Cam19] for a parallel implementation of Laguerre's method.

Transforming Newton's method into a splitting algorithm requires choosing a set of starting points for which the dynamics will visit every root. The naive approach of findind one root at a time and factoring it out is both unpractical and numerically

unstable (see §3.1). Choosing starting points at random may miss some roots entirely.

For a given $z_0 \in \mathbb{C}$ for which the sequence converges to z_*, the number of steps required before the sequence enters the bi-exponential regime *i.e.* $\min\{n\,;\, z_n \in \mathrm{D}(z_*, \rho)\}$ with ρ as in (16) may be astronomical. Occasional wild excursions from the vicinity of critical points of P towards ∞ are not the main problem: those trajectories are easily identified and abandoned. The less understood regime happens when the sequence crosses the mid-range $\mathrm{D}(0,r)\setminus \mathrm{D}(z_*,\rho)$ where the sequence may almost stall if the Newton's bassin forms a narrow gorge, as described in [HSS01]; see also Figure 2. Any theoretical advance on this matter like [BAS16] are of interest to improve the bounds on the complexity of Newton's method as a global splitting algorithm.

The stoping criterion is another critical issue: P. Henrici [Hen74, Chap. 6] states that one may stop the Newton method when its steps get smaller then ε/d where ε is the precision sought after, in which case the proximity of a root can be proved. In our algorithm, we stopped much earlier and relied instead on an a-posteriori analysis, performed in a higher precision and using disk arithmetic to prove that the sequence has entered the quadratic convergence region (see §5).

Aberth-Ehrlich method [Abe73] is a generalisation of Newton's method. It attempts to find all the roots simultaneously by computing a fixed point for $z_n = (z_{n,j}) \in \mathbb{C}^d$:

$$z_{n+1,j} = z_{n,j} - \frac{1}{\frac{P'(z_{n,j})}{P(z_{n,j})} - \sum\limits_{k\neq j} \frac{1}{z_{n,j}-z_{n,k}}}.$$

The idea is that each root repels the others with an electrostatic potential, which prevents the Newton's dynamics to converge multiple times on a single root while missing another one completely. The components of the starting vector $z_0 \in \mathbb{C}^d$ are chosen within $\mathrm{D}(0,r)$ and presumed to lack any symmetries that would accidentally match a symmetry of the set of roots, which could prevent convergence.

The Aberth method is at the heart of the implementation of `MPSolve`, which is the reference software for splitting polynomials using arbitrary precision arithmetic. A massively parallel implementation [GSKC16] of this algorithm on multiple GPUs can run up to a few millions of roots.

At each step, computing the electrostatic potential requires $O(d^2)$ operations, while the evaluation of all $P'(z_{n,j})/P(z_{n,j})$ costs $O(d\,\mathrm{V}_d)$. The typical number of steps to reach a given approximation depends strongly on the choice of initial points [Gug86], [Fra89], [Bin96]; in practice, it takes at least $O(d)$ steps, with an overhead of $O(d\log d)$ operations to compute a "good" initial vector $z_0 \in \mathbb{C}^d$. The overall cost of Aberth's method is thus $O(d^3)$ operations regardless of the actual value of V_d.

In practice, the Aberth method has good global convergence properties. However, a theoretical foundation for this global convergence remains an open problem.

Weierstrass method (also known as Durand–Kerner) is the following variation:

$$z_{n+1,j} = z_{n,j} - \frac{P(z_{n,j})}{\prod\limits_{k\neq j}(z_{n,j}-z_{n,k})}.$$

On a theoretical level, it was recently proven [RSS20], [KI04] that the Weierstrass method may fail generically, *i.e.* on an open set of polynomials for an open set of intialization points. This is a sharp reminder that one must pay close attention to the bassin of attraction of each numerical method.

3.5 The algorithm of Hubbard, Schleicher and Sutherland

The global structure of the bassins of attraction of Newton's method is quite intricate. It has been extensively studied in [Sut89], [HSS01], [SS17], where a remarkable universal algorithm for splitting P is explained. Let us consider a root z_* of P and denote by $\mathcal{A}(z_*) \subset \overline{\mathbb{C}}$ the basin of attraction for Newton's method.

The connected component of z_* in $\mathcal{A}(z_*)$ is called the immediate basin of attraction and is denoted by $\mathcal{U}(z_*)$. It admits as many *access* to infinity (*i.e.* homotopy classes of simple arcs connecting z_* to ∞) as the number of critical points of N_P within it, counted with multiplicity (including the root itself as critical point if z_* is a multiple root). The *girth* (or *channel width*) at $z \in \mathcal{U}(z_*)$ is the length of the connected component of z in

$$\{z' \in \mathcal{U}(z_*)\,;\, |z'| = |z|\}.$$

The girths admits a universal bound from below, which implies that taking about $2.47d^{3/2}$ points uniformly along a wide enough circle ensures that there is at least one point per basin of attraction. This property turns Newton's method into a trivially parallelizable global splitting algorithm.

A more involved analysis based on the conformal modulus of the access channels (roughly speaking: the ratio between the girth of the channel to the amplitude of one Newton step from that point) allows both a reduction in the number of points involved, and better estimates. A universal starting mesh, given in [HSS01], is composed of about $\alpha \simeq 4.16 d \log d$ points taken on $\beta \simeq 0.27 \log d$ concentric circles.

Theorem 3 (Splitting algorithm of [HSS01])**.** *Given a polynomial $P(z)$ of degree d whose roots are contained in $\mathrm{D}(0,r)$ and integers $\alpha \geq 4.16 d \log d$, $\beta \geq 0.27 \log d$, the mesh*

$$\mathbb{M}_r(d) = \bigcup_{1 \leq \nu \leq \beta} r(1+\sqrt{2}) \left(1 - \frac{1}{d}\right)^{\frac{2\nu-1}{4\beta}} \mathbb{U}_\alpha$$

contains a point in each bassin of attraction of the roots of P, for Newton's method. Here, $\mathbb{U}_\alpha = \left\{e^{2ik\pi/\alpha}\,;\, 0 \leq k < \alpha\right\}$ denotes the roots of unity of order α.

Overall, $\#\mathbb{M}_r(d) \simeq 1.11 d \log^2 d$. For $d \simeq 2^{32}$, the mesh is build upon $\beta = 6$ circles. The computational improvement over a one-circle approach, which requires $O(d^{3/2})$ starting points, becomes advantageous when $d \geq 2^{14}$ (see Figure 5).

3.6 Practical considerations regarding the algorithm of [HSS01]

In this subsection, we discuss some practical aspects of the implementation of Theorem 3, seen as a splitting algorithm, that will lead (for the polynomials p_n and $q_{\ell,n}$) to our improved algorithm presented in Section 4.

The overall complexity of J.H. Hubbard, D. Schleicher and S. Sutherland splitting algorithm is $\alpha\beta\gamma$ Newton steps (*i.e.* $\alpha\beta\gamma V_d$ arithmetic operations), where $\alpha\beta = \#\mathbb{M}_r(d)$ is the number of starting points and γ is the average number of Newton iterations that are actually necessary to compute the roots with a precision ε. The question of estimating γ is two-fold. It obviously depends on the minimal separation of the roots of $P(z)$, which dictates the precision ε of the (final) computations. In [HSS01], no upper bound on γ is given. However, a practical number of iterates is recommended:

$$\gamma \simeq d \log\left(\frac{r}{\varepsilon}\right) \tag{19}$$

assuming that roots are simple. On the other hand, γ is also profoundly impacted by the mid-range dynamics of the Newton map, *i.e.* the region $\mathrm{D}(0, r(1+\sqrt{2}))\setminus \mathrm{D}(z_*, \rho)$ with ρ as in (16), where the method can spend a substantial amount of time. The key observation that leads to our algorithm is that, in that region, Newton's method appears to be computing level lines over and over in a very inefficient way (see Figure 2).

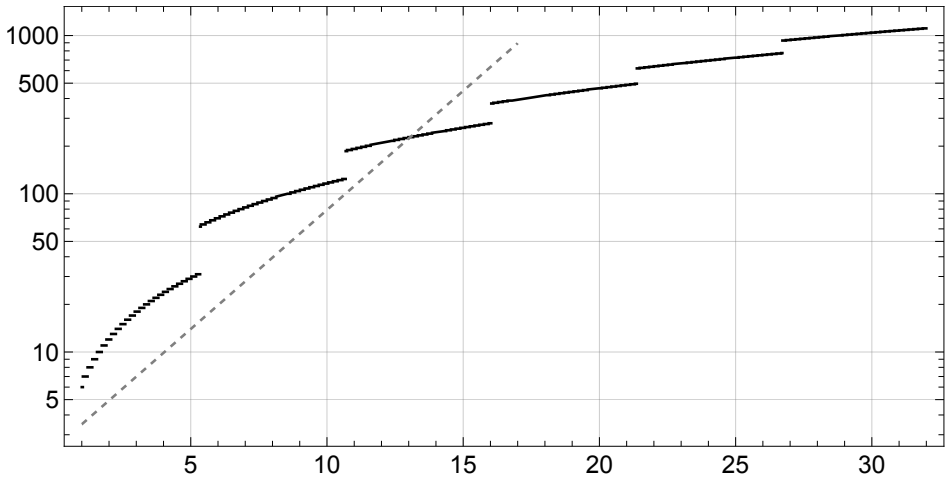


Figure 5: Gridpoints-per-root ratio for $\mathbb{M}_r(d)$ as a function of $\log_2(d)$. Comparison with the one-circle grid (dashed line).

3.6.1 On the scalability of the number of gridpoints per root

The algorithm of [HSS01] is remarkably efficient when $d \leq 2^{20}$ *i.e.* for polynomials of degree up to a few millions. However, in the giga- and tera-scale, the growth of the gridpoint-per-root ratio, *i.e.* $\alpha\beta/d$, may add a substantial overhead to the cost of the computation (see Figure 5). In other terms, having the certainty to catch all the roots

comes at a cost of at least 500 redundant finds for each root, on average. Moreover, eliminating those redundant finds adds, in practice, additional search and sorting costs. This objection is, however, not completely fair: in practice, one can start the search on a dyadic thinning of $\mathbb{M}_r(d)$ and gradually step up to using the full mesh if roots are still missing after each pass (see [HSS01, §9]).

Remark 3. *If P is known to only have real roots, then the mesh $\mathbb{M}_r(d)$ can be reduced to $1.3d$ points on a single semi-circle, which gives a constant gridpoints-per-root ratio.*

3.6.2 On root separation

Let $\mathcal{Z} = P^{-1}(0)$. For any $z \in \mathcal{Z}$, let us introduce

$$\delta_P(z) = \min\left\{|z - z'|\,;\, z' \in \mathcal{Z} \text{ and } z' \neq z\right\}. \tag{20}$$

In order to be able to distinguish the root z from its closest neighbor, it is necessary to compute it with a precision $\varepsilon < \delta_P(z)/2$. The minimal separation between the roots of the polynomial P is

$$\Delta_P = \min\left\{\delta_P(z)\,;\, z \in \mathcal{Z}\right\}, \tag{21}$$

For example (see Figure 15), for hyperbolic polynomials, one has $\Delta_{p_n} \propto 4^{-n}$ which means that, lacking any further information on the local fluctuations of $\delta_{p_n}(z)$, one should request a precision ε of order $O(4^{-n})$ for all the roots, which, according to (19) requires $\gamma = O(n2^n)$ iterations of the Newton map. The cardinality of the starting mesh is at least $d = 2^{n-1}$ and may skyrocket up to $\#\mathbb{M}_2(d) = O(n^2 2^n)$ should the finest mesh be required. The total number of Newton steps thus ranges from $O(n2^{2n})$ to $O(n^3 2^{2n})$. As the evaluation of p_n requires $V_d = O(n)$ arithmetic operations, applying [HSS01] to split p_n requires $O(n^\kappa 2^{2n})$ arithmetic operations with $2 \leq \kappa \leq 4$.

We have no reason to believe that p_{41} would behave especially badly. However, the previous discussion gives a range of 10^{27} to 10^{31} arithmetic operations, which means that, even if we disregard the need to compute at a precision much higher than 64 bits and forget all the extra operations required to handle the large volume of data, it would take, optimistically, 31 years to split p_{41} on a exa-scale computer like *Frontier*. This optimistic estimate is astronomical and marks the splitting of p_{41} as a problem clearly out of reach of the current state of the art.

Remark 4. *The root separation of p_n will be analysed in details in §6.3.2 (see Figure 16), which would improve the overall cost if the precision ε was adjusted dynamically.*

3.6.3 On the mid-range dynamics of the Newton map

A pertinent quantity for the analysis of the Newton map is the number of Newton steps that are necessary to bridge the level set A to the level set B (with $A > B$):

$$\gamma_{A,B}(P, z_0) = \min\{k\,;\, |P(z_k)| \leq B\} - \max\{k\,;\, |P(z_k)| \geq A\}. \tag{22}$$

For each root z_*, the dynamics of the Newton map develops in two stages. The first stage (called mid-range) connects the starting mesh $\mathbb{M}_r(d)$ to the edge of $\mathrm{D}(z_*, \rho)$, which requires $\gamma_1 \simeq \gamma_{A,B}(P, z_0)$ Newton steps where $A = |P(z_0)|$ and $B = \max\{|P(z)|\,;\, z \in \mathrm{D}(z_*, \rho)\}$ with ρ as in (16). In practice, $\gamma_1 \geq O(d)$ because, according to [HSS01, Lemma 3], Newton steps satisfy

$$\frac{|z| - r}{d} < |N_P(z) - z| < \frac{|z| + r}{d} \tag{23}$$

for $|z| \geq r > 0$. A second stage connects the edge of $\mathrm{D}(z_*, \rho)$ to $\mathrm{D}(z_*, \varepsilon)$ where the desired precision is reached. As it happens entirely in the quadratic regime (16), this phase requires and additional $\gamma_2 = O(\log \frac{\rho}{\varepsilon})$ refining steps. Overall $\gamma = \gamma_1 + \gamma_2$.

Remark 5. *As discussed in [HSS01, §9], starting instead from a rescaled mesh or radius λr for some $\lambda > 1$ would allow one to slightly decrease the upper bound on the cardinality of the mesh if λ is chosen close to 1, but it would dramatically increase γ_1 by $O(d \log \lambda)$.*

Computing $\gamma_{A,B}$ precisely in full generality seems hopeless (see however [BAS16]). However, one may experiment along the real line to get a feeling of the orders of magnitude. For the hyperbolic polynomials p_n, the Newton dynamics starting from the point

$$z_0 = \frac{1}{4} + \theta_n \in \mathbb{R}_+ \quad \text{such that} \quad p_n\left(\frac{1}{4} + \theta_n\right) = 5$$

always converges to $z_* = 0$. The first stage of the dynamics takes

$$\gamma_1 = \inf\{k \in \mathbb{N}\,;\, z_k \in \mathrm{D}(0, 1/4)\}$$

steps because for $k \geq \gamma_1$, the convergence of the sequence $(z_k)_{k\in\mathbb{N}}$ becomes quadratic. We checked numerically that $\gamma_1 \leq 6$ for any $k \leq 38000$. On the other hand, starting another sequence of Newton iterations from $\widetilde{z}_0 = 1$ (as would be the case if we start from $\mathbb{M}_r(2^{n-1})$), one would get

$$\widetilde{\gamma}_1 = \inf\{n \in \mathbb{N}\,;\, p_k(\widetilde{z}_n) \leq 5\} \geq O(2^k)$$

because each Newton step is of order

$$|p_k(\widetilde{z}_n)/p_k'(\widetilde{z}_n)| \leq \frac{p_k(1)}{p_k'(\frac{1}{4} + \theta_k)}$$

This simple observation suggests that, instead of starting from a universal and too far mesh $\mathbb{M}_r(2^{n-1})$, it may be more appropriate to start from a mesh along a *level line*

$$|p_n(z)| = L.$$

This idea is the key to the new algorithm that we describe in the next section.

4 A new splitting algorithm based on level-lines

In this section, we expose our new splitting algorithm at the general level, and illustrate it on the search for hyperbolic centers and Misiurewicz points of $\mathcal{M}$. Again, throughout this section P denotes a polynomial in $\mathbb{C}[z]$ of degree $d = \deg P$ whose roots $\mathcal{Z} = P^{-1}(0)$ are located within the disk $\mathrm{D}(0, r)$. The set of critical points is denoted by $\mathrm{Crit}\, P$. For simplicity, we will also assume that P is monic.

Brute force is not a viable option when $d \gtrsim 10^{12}$. For example, one observes that the minimal separation between roots of p_k changes dramatically with the position in $\mathcal{M}$ and that only a few roots contribute to the realisation of the overall minimum (see figure 16). The corresponding Newton bassins are narrow channels. As illustrated in the previous section, using a uniform starting grid fine enough to capture all of those narrow channels is an algorithmic dead-end.

4.1 General principle

The solution we propose is that of an adaptative mesh refinement. Trying to reverse-engineer the dynamics of the Newton map on the fly to guess where to add points seems mostly out of reach and has only been attempted by [RSS17]-[Sch23] who use a clever geometric estimator of the deformation of the trajectories for that purpose; however, if a few roots end up being missed, it is not clear how to refine the computation.

What we need instead is a systematic way to put more grid points around regions of high root density and spend less time visiting the regions of lower root density. We want to parametrise a curve encircling the roots of P such that uniform subdivisions of its parameter induce the desired adaptative mesh refinement.

The answer is given by the interaction between the two ordinary differential equations that rule Newton's flow and the level lines. *Newton's flow* is defined by the ODE:

$$\zeta'(t) = -\frac{P(\zeta(t))}{P'(\zeta(t))} \tag{24}$$

The evolution is iso-angle (*i.e.* $\arg P$ is constant along trajectories) and the modulus is an exponentially decaying Lyapunov function because any solution satisfies:

$$P(\zeta(t)) = e^{-(t-t_0)} P(\zeta(t_0)). \tag{25}$$

The stationary solutions are the roots. We will call *Newton's rays* or *iso-angle rays* the non-stationary solutions of the Newton flow. The non-critical iso-angles, *i.e.*

$$\arg P(\zeta(t_0)) \notin \arg\left[P(\mathrm{Crit}\, P)\backslash\{0\}\right] \tag{26}$$

stem global solutions, both forwards and backwards. Forwards, non-stationary maximal trajectories on $[t_0, T^*)$ converge to the roots if the trajectory is global *i.e.* $T^* = +\infty$, or to critical points if $T^* < +\infty$. Note that, when $T^* < +\infty$, the argument of the critical

value at the end-point must match the iso-angle of the trajectory; also, the end-point cannot be a multiple root because (25) implies that

$$\lim_{t\to T^*} P(\zeta(t)) = e^{-(T^*-t_0)}P(\zeta(t_0)) \neq 0.$$

Backwards, maximal trajectories on $(T_*, t_0]$ either go to $\infty \in \overline{\mathbb{C}}$ if $T_* = -\infty$ or to non-root critical points with the same critical iso-angle if $|T_*| < \infty$. Bounded maximal trajectories are possible between two critical points whose values share the same argument. Note that 0 is a critical value if and only if P admits multiple roots. In that case, if $\zeta(t)$ converges to a root of multiplicity μ, L'Hopital's rule ensures that

$$\zeta'(t) \underset{t\to+\infty}{\sim} -\frac{P^{(\mu-1)}(\zeta(t))}{P^{(\mu)}(\zeta(t))}$$

so, locally, the profile of Newton's flow is radial and coincides with that of a simple root of $P^{(\mu-1)}$. The properties of Newton's flow where used in [AMV23] to produce a short proof of the fundamental Theorem of Algebra.

Level lines $|P(z)| = L$ are also caracterized by an ODE, namely:

$$\lambda'(t) = i\frac{P(\lambda(t))}{P'(\lambda(t))}. \tag{27}$$

The solutions satisfy

$$P(\lambda(t)) = e^{it}P(\lambda(0)). \tag{28}$$

By construction, the level-lines are orthogonal to iso-angles. Again, stationary solutions are the roots of P. The modulus $|P(\lambda(t))|$ is constant along each level-line; in particular, the level-lines are bounded. If $|P(\lambda(t))| > \max|P(\mathrm{Crit}(P))|$, there is only one maximal solution, which is a Jordan curve (as the image of a circle by the Riemann map of the outside). Below the maximal critical value, there are multiple connected components that are either compact or join two (possibly identical) critical points. The function $\lambda(t)$ is $2\ell\pi$-periodic where $\ell \leq d = \deg P$ because $P(\lambda(2k\pi)) = P(\lambda(0))$ for any $k \in \mathbb{Z}$ and P can take this value at most d distinct times; once the value loops, the unicity in the Cauchy-Lipschitz theorem for the first-order ODE (27) ensures periodicity. For a level line of modulus large enough, d is the smallest period.

Theorem 4. *Given a monic polynomial P of degree d, if $z_0 \in \mathbb{C}$ satisfies*

$$|P(z_0)| > \max|P(\mathrm{Crit}(P))| \qquad \textit{and} \qquad \arg P(z_0) \notin \arg[P(\mathrm{Crit}(P))\backslash\{0\}], \tag{29}$$

then the points $z_k = \lambda(t_0 + 2k\pi)$ for $0 \leq k < d$ obtained by solving (27) with $\lambda(t_0) = z_0$ are distinct. Each z_k is associated to a global forward Newton flow (24), denoted by $\zeta_k(t)$. The points $z_k^* = \lim_{t\to+\infty} \zeta_k(t)$ are all the roots of P and this enumeration is consistent with the multiplicities, i.e.

$$P(z) = (z - z_0^*)(z - z_1^*)\dots(z - z_d^*). \tag{30}$$

Proof. From the discussion above, we know that the starting points $\lambda(t_0 + 2k\pi)$ for $0 \leq k < d$ such that $\arg P(\lambda(t_0)) \notin \arg[P(\mathrm{Crit}(P))\backslash\{0\}]$ have a global forward Newton flow and that each of these flows converges to a root of P. The d starting points are distinct because $|P(\lambda(t_0))| > \max|P(\mathrm{Crit}(P))|$ so the level line has only one connected component. The question is wether one could be missing a root. Near each root z_*, the polynomial P takes all possible arguments exactly as many times as the multiplicity because $P(z) \sim \alpha(z - z_*)^m$ as $z \to z_*$, for $\alpha \in \mathbb{C}^*$ and $m \in \mathbb{N}^*$. If we missed a root z_*, we consider a nearby point where $\arg P(z) = \arg P(\lambda(t_0))$. The Newton ray from z is global both forwards and backwards and escapes (backwards) at infinity. Thus, this Newton ray intersects the level line of modulus $|P(\lambda(t_0))|$ and, as the angle is invariant, it is one of the $\lambda(t_0 + 2k\pi)$ points. Therefore, no root could have been missed. The same argument shows that no multiplicity can be misrepresented, by following back the m Newton rays near z_* whose angle is $\arg P(\lambda(t_0))$. □

If one chooses d distinct values $(t_j)_{0\leq j<d}$ in $[0, 2\pi)$, then at least one of them must satisfy $\arg P(\lambda(t_j)) \notin \arg[P(\mathrm{Crit}(P))\backslash\{0\}]$ because $\arg P(\lambda(t_j)) = \arg P(\lambda(t_0)) + t_0 - t_j \mod 2\pi$ and there are, at most, $d-1$ distinct critical values. We have therefore a naive way of choosing $O(d^2)$ starting points $(\lambda(t_j+2k\pi))_{0\leq j,k<d}$ whose Newton trajectories are guarantied to reach all the roots. Generically however, the number of starting point drops to $O(d)$ as it is very unlikely that more than a few choices will hit critical arguments. For example, in practice, we were able to split the polynomials p_n and $q_{\ell,n}$ by Newton's method using only $4d$ starting points of iso-angles 0, $\pi/2$, π and $3\pi/2$.

As illustrated on Figure 2, the next advantage of using iso-angle rays and level lines instead of using unstructured parallel iterations of the Newton map, is that one can perform most of the descend using a lean mesh, and densify it at a lowest level with no risk of missing iso-angle rays.

For $\zeta_0 \in \mathbb{C}$ such that $|\zeta_0| > \max|P(\mathrm{Crit}(P))|$ and $N \in \mathbb{N}^*$, let us consider the mesh

$$\mathbb{L}_{\zeta_0,N}(P) = \{\lambda(2kd\pi/N)\,;\, 0 \leq k < N\} \tag{31}$$

where λ is the $2d\pi$-periodic maximal solution of (27) defined by a choice[6] of $\lambda(0)$ as one arbitrary root of $P(\lambda(0)) = \zeta_0$. In the rest of this section, we assume that this choice is made consistently, even though it does not interfere with the set itself because λ is periodic. The formula (28) implies that:

$$P(\lambda(2kd\pi/N)) = e^{2ikd\pi/N}\zeta_0. \tag{32}$$

In particular, the mesh $\mathbb{L}_{\zeta_0,N}$ is a finite subset of cardinality N of the (smooth) level curve defined by $|P(z)| = |\zeta_0|$. It is called a *discrete level set* of order N. As

$$\mathbb{L}_{\zeta_0,N}(P) \subset \mathbb{L}_{\zeta_0,2N}(P),$$

[6]To lift the ambiguity regarding that choice, one can also denote the level set by $\mathbb{L}^{\bullet}_{z_0,N}(P)$ as the level set that *starts* at $\lambda(0) = z_0 \in \mathbb{C}$; it is the level of $\zeta_0 = P(z_0)$, *i.e.* $\mathbb{L}^{\bullet}_{z_0,N}(P) = \mathbb{L}_{\zeta_0,N}(P)$.

each mesh $\mathbb{L}_{\zeta_0,2^m}(P)$ is a refinement of $\mathbb{L}_{\zeta_0,2^{m-1}}(P)$. The mesh $\mathbb{L}_{\zeta_0,N}(P)$ is naturally self-refining. Increasing the value of N creates a finer uniform mesh on the parameter side (roots of unity of higher order). However, on the preimage side of P, the density of the mesh increases naturally around points of higher root density, as illustrated on Figure 6. Note that the previous considerations on the solutions of (27) imply that

$$P^{-1}(\zeta_0) = \mathbb{L}_{\zeta_0,\deg P}(P)$$

as long as $|\zeta_0| > \max|P(\mathrm{Crit}(P))|$.

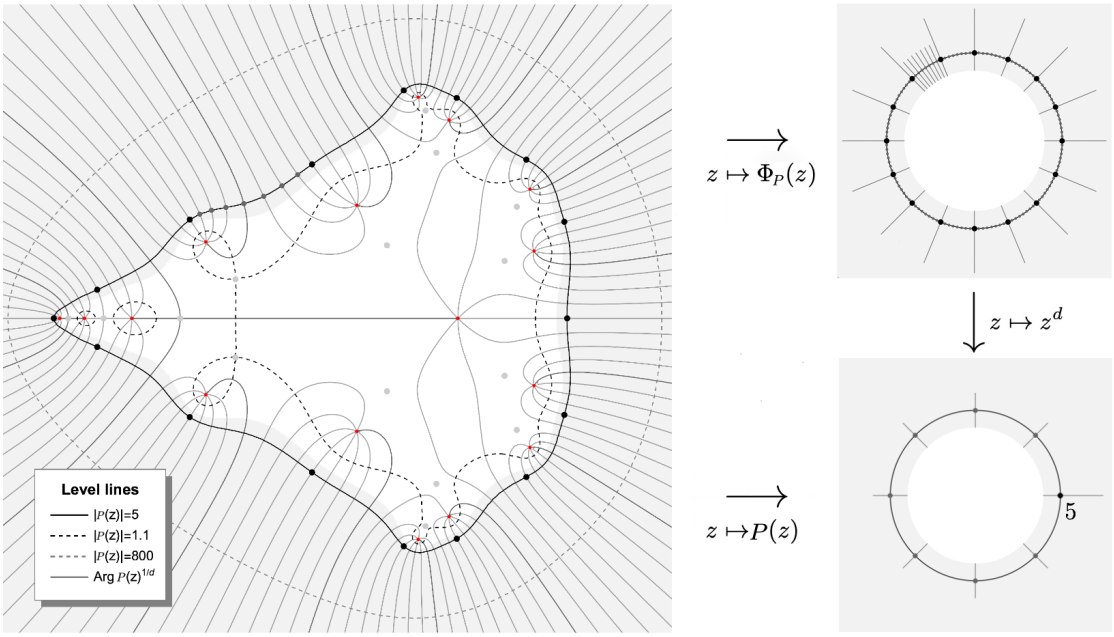


Figure 6: Initial mesh $\mathbb{L}_{\zeta_0,d}(p_5)$ as black points and part of $\mathbb{L}_{\zeta_0,8d}(p_5)$ with $d = 16$ and $\zeta_0 = 5$. Two other level lines are depicted along with Newton's flow (iso-angles). The map $\Phi_P(z) = P(z)^{1/d}$ maps large level lines to circles (top right) while P folds the level line ζ_0 onto itself d times (bottom right). This figure is a commutative diagram.

Let us assume for now that one is able to compute $\mathbb{L}_{\zeta_0,N}(P)$ efficiently. Then, the (theoretical) algorithm consists in iterating either Newton's flow or the Newton map from each point of $\mathbb{L}_{\zeta_0,N}(P)$ until roots are found up to the precision desired. Obviously, this step is highly parallelizable. Duplicates are eliminated from the list and if some roots are missing, N can be increased.

The strategy for splitting P can be formalized as follows. The constant c_L is chosen in accordance with Proposition 4.1. In practice, $c_L = 4$.

Algorithm $\mathcal{S}(P)$: Splitting algorithm for a polynomial P (sequential version)

Data: $\lambda_0 > \max|P(\mathrm{Crit}(P))|$ and η level of quadratic convergence for N_P.

```
Compute the discrete level line 𝕄 = 𝕃_{λ0,c_L d}(P)                    /* O(d) */
for each z ∈ 𝕄 do
    Iterate time-1-flow up to level η                  /* O(T_flow × log λ0/η) */
    Iterate N_P up to desired precision ε                  /* O(log|log ε|) */
    Add root to appropriate data structure R                          /* O(1) */
```

The discrete level line $\mathbb{L}_{\zeta_0,N}(P)$ is computed recursively using the following result. One defines the Newton map with target $\zeta_0 e^{i\vartheta}$ as the limit, if it exists:

$$L(P; z_0, \zeta_0, \vartheta) = \lim_{n\to\infty} N^n_{P-\zeta_0 e^{i\vartheta}}(z_0) \tag{33}$$

where N_Q is the Newton map (15) of a polynomial Q and powers denote, as usual, composition. By construction, $P(L(P; z_0, \zeta_0, \vartheta)) = \zeta e^{i\vartheta}$.

Proposition 4.1. *Given z_0 such that $|P(z_0)| > \max|P(\mathrm{Crit}(P))|$, an arbitrary integer $N \in \mathbb{N}^*$ there exists $M \in \mathbb{N}^*$ large enough such that the finite sequence*

$$z_{k+1} = L\left(P; z_k, P(z_k), \tfrac{2\pi d}{MN}\right) \tag{34}$$

satisfies $\mathbb{L}_{P(z_0),N} = \{z_0, z_M, z_{2M}, \ldots, z_{(N-1)M}\}$.

Remark 6. *In practice, $\mathbb{L}_{\zeta_0,N}(P) = \{\tilde{z}_0, \ldots, \tilde{z}_{N-1}\}$ where $\tilde{z}_k = z_{kM}$ where the auxiliary finite sequence $(z_{kM+j})_{j\in[\![0,M]\!]}$ is defined recursively from $\tilde{z}_k$ such that*

$$P(z_{kM+j}) = \zeta_0 e^{2i\pi d\,\frac{kM+j}{NM}}.$$

When $N = d$, the phase shift is simpler. The phase of $P(z)$ changes $d = \deg P$ times along $|P(z)| = \lambda$; each time the phase advances by 2π (one turn), we have a new point of $\mathbb{L}_{\zeta_0,d}(P)$. Each discrete turn is completed with M intermediary steps, as illustrated on Figure 6. The value M is called the number of points per turn for the computation of $\mathbb{L}_{\zeta_0,d}(P)$. When N divides d, roots of $P(z) = \zeta_0$ are being skipped.

Proof. Let $C = \max|P(\mathrm{Crit}(P))|$. Applying the same argument as in [Jun85], one can claim the following: for $\zeta \in \mathbb{C}$ such that $|\zeta| > C$ there is unique analytic determination of $\Phi_P(z) = P(z)^{1/d}$ which is determined by the constraint $\Phi_P(z) \sim z$ as $|z| \to \infty$. Moreover, the map Φ_P is univalent on $\{z \in \overline{\mathbb{C}}\,;\, |P(z)| > C\}$. Choosing a guard $\gamma > C$ and a number of intermediary points $M \geq \frac{2\pi\gamma}{|\zeta|-C}|\zeta|$ ensures

$$|P(z_{kM+j})| - C \geq \gamma|P(z_{kM+j}) - P(z_{kM+j+1})|. \tag{35}$$

The rigidity provided by Kobe's Theorem 12 ensures that one captures the right root. □

Heuristically, Newton's descend can be split in two phases. In the first phase, the discrete version (iterating N_P) is roughly equivalent to following the iso-angle rays in time 1. An exact upper-bound of the number of steps is available for the continuous flow. We call T_{flow} the arithmetic cost of following the continuous flow in time 1 and factor it out.

Proposition 4.2. *Given a point z_0 such that $|P(z_0)| = \lambda_1$ and $\lambda_2 < \lambda_1$, the Newton flow $\zeta(t)$ of P defined by (24) with $\zeta(0) = z_0$ satisfies*

$$|P(\zeta(t))| \leq \lambda_2 \qquad \text{if and only if} \qquad t \geq \log \frac{\lambda_1}{\lambda_2}.$$

In other terms, the number of iterations of the time 1 Newton flow that connect two level lines is logarithmic in the ratio of the levels.

Proof. According to (25), each time 1 iteration of Newton's flow divides the modulus of the value of the polynomial by e. □

In the second phase (assuming simple roots), N_P converges quadratically.

Proposition 4.3. *If $P^{-1}(D(0,\eta))$ in included in the union of the quadratic bassins of the Newton's method, it takes $O(\log\log\eta/\varepsilon)$ Newton steps to go from the level line η to the final desired precision ε.*

The appropriate data structure that allows sorting and insertions of d points with total complexity of only $O(d)$ will be presented in Section 6.

4.2 Specifics for polynomials related to $\mathcal{M}$

In full generality, $\mathcal{S}(P)$ is not yet ready to rival the algorithm of [HSS01]. However, in practice, our strategy behaves extremely well (see Section 6).

In our implementation, the outmost level line is $\zeta_0 = 5$ for p_n and $\zeta_0 = 100$ for $q_{\ell,n}$. For the smallest degrees, there is no need for parallelism and it is not necessary to refine the level curve: one applies simply $\mathcal{S}(P)$ with $4d$ points, *i.e.* $c_L = 4$.

To split p_n and $q_{\ell,n}$ for large n, a massively parallel implementation is welcome and can be formalized as follows. Upper bounds on the arithmetic complexity are given as commentaries. The mention *para* indicates that the operation can be performed in a trivially parallel fashion (*i.e.* as J independent jobs) on multiple computing units ; on the other hand, the mention *mono* is a step that should be performed on a single computing unit.

One chooses a constant c_N that will cap the number of Newton iterates to $c_N \log d$. In practice, $c_N = 1$ for p_n or $c_N = 2.6$ for $q_{\ell,n}$.

Algorithm $\mathcal{S}_{/\!/}$: Splitting algorithm for p_n and $q_{\ell,n}$ (parallel & certified version)

begin compute level lines
  `coarse level line` $\mathbb{L}_{\lambda_0,N_0}(P)$ `/* O(1) mono */`
  `refine level line to` $\mathbb{L}_{\lambda_0,N_1}(P) = \bigcup \mathbb{M}_j$ `/* O(d) para */`
    `with` $N_1 \in N_0 \times 2^{\mathbb{N}}$ `such that` $N_1 = c_L d$
begin root finding
  for each *parallel job* $j \in \{1, \dots, J\}$ **do**
    for each $z \in \mathbb{M}_j$ **do**
      `iterate` N_P `up to` $\max\{20; c_N \log d\}$ `/* O(log d) para */`
    `sort roots` R_j `in appropriate data structure` `/* O(1) para */`
  `merge roots from all jobs` $R = \bigcup R_j$ `/* O(d/J) mono */`
begin certification
  for each $z \in R$ **do**
    `check` $P(z) = 0$ `using disk arithmetic` `/* O(log d) para */`

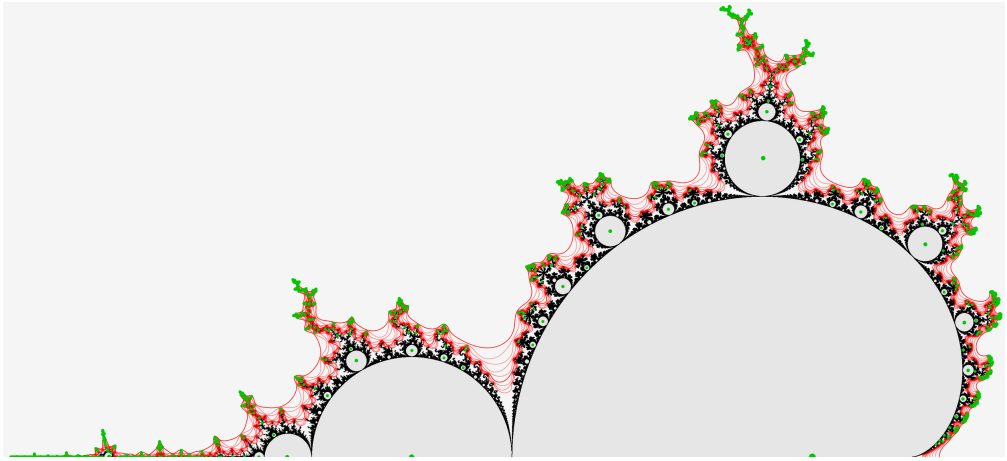


Figure 7: Level sets of p_k for $10 \le k \le 18$ (red) and hyperbolic centers $\mathrm{Hyp}(n)$ for $n \le 12$ (green). The level lines $\mathbb{L}_{\lambda,N}$ tend to self-refine around regions where $\arg P(z)$ cycles most, which reflects a higher intrication of the bassins of attraction for the Newton flow and the Newton map.

Let us go over each step in details and justify the bounds on the arithmetic complexity claimed above.

The mesh refinement is possible because the p_n and $q_{\ell,n}$ are defined recursively (see below). Each point $\mathbb{L}_{\lambda_0,N_0}(P)$ can then be used as a starting point of an independent job so one can choose $J \le N_0$. By construction, there are $N_1/N_0 - 1$ points of $\mathbb{L}_{\lambda_0,N_1}(P)$

between each pair of consecutive points of $\mathbb{L}_{\lambda_0,N_0}(P)$. According to Theorem 4, it means that the algorithm will produce $\#R_j \leq d/J$ roots per job.

For large degrees (typically $n \geq 17$), the coarse level set corresponds to $N_0 = 2^{15}$ which means that 16 385 starting points such that $|P(z)| = \lambda_0$ are computed in the upper plane $\operatorname{Im} z \geq 0$, with $\arg P(z) = 2k\pi/N_0$ and $k \in [\![0, N_0]\!]$. For the intermediary periods, we found it more practical to group consecutive jobs together in order to ensure an approximately constant size for $\#R_j$ (and thus consistent file sizes across the project).

Let us now explain how $\mathbb{L}_{\lambda_0,c_L d}(P)$ can be computed efficiently using a parallel algorithm. As the cardinality of this mesh is $c_L d$, the huge volume of data prohibits to compute the mesh ahead of time or to store it on disk when d reaches the tera-scale. The pertinent subsection of the mesh has to be generated on the fly by each computing process.

For the Mandelbrot set, one uses the fact that the polynomials $(p_k)_{k\geq 3}$ form a nested family whose level lines encircle $\mathcal{M}$ tightly (see Figure 7). One computes an initial mesh $\mathbb{L}_{\lambda_0,2^{15}}(p_{16})$. Next, one observes that

$$p_{17}(z_j) = p_{16}(z_j)^2 + z_j = \lambda_0^2 + z_j.$$

Using the guard γ as in (35), we can find z'_j close to z_j such that $p_{17}(z'_j) = \lambda_0^2$. This means that $z'_j \in \mathbb{L}_{\lambda_0^2,2^{15}}(p_{16})$ with the correct index. Iterating this method, one can then use z'_j as starting points to solve iteratively $p_{17}(z) = r_k$ where the radius r_k is gradually reduced from λ_0^2 down to λ_0. In the end, one obtains $\mathbb{L}_{\lambda_0,2^{15}}(p_{17})$. In a similar fashion, one can construct the mesh $\mathbb{L}_{\lambda_0,2^{15}}(p_{k+1})$ from $\mathbb{L}_{\lambda_0,2^{15}}(p_k)$ in negligible time. This method is the key to the parallelization as each point of $\mathbb{L}_{\lambda_0,2^{15}}(p_k)$ is the seed of a parallel task.

The guard γ in (the proof of) Proposition 4.1 is $\gamma = 6.1$ because of the following statement, which is based on the fact that the highest critical value (in module) of p_n is reached along the real axis, at the left-most tip of the Mandelbrot set and is asymptotically equal to 2.

Theorem 5 ([Jun85])**.** *There exists a sequence of analytic maps $\Phi_k : \mathcal{M}^c \to \overline{\mathbb{C}}$ such that*

$$\forall k \geq 3, \qquad \Phi_k(c)^{d_k} = p_k(c) \qquad \text{and} \qquad \Phi_k(c) \underset{|c|\to\infty}{\sim} c \tag{36}$$

with $d_k = 2^{k-1}$. The map Φ_k is one-to-one from $U_k = \{z \in \overline{\mathbb{C}}\,;\, |p_{k+1}(c)| > 2\}$ onto the outer disk $\{z \in \overline{\mathbb{C}}\,;\, |z| > 2^{2^{-k}}\}$. The sequence converges uniformly on compact sets to the Riemann map $\mathcal{M}^c \cup \{\infty\} \to \bar{\mathbb{C}}\backslash\overline{\mathbb{D}(0,1)}^c$.

Remark 7. *The Riemann map is given by $\Phi(c) = \varphi_c(c)$ where φ_c is the map*

$$\varphi_c(z) = z \prod_{n=0}^{\infty} \left(1 + \frac{c}{f_c^n(z)^2}\right)^{2^{-n-1}} = \lim_{n\to\infty} f_c^n(z)^{2^{-n}}$$

that conjugates the dynamics on $f_c^{-1}(\overline{\mathbb{C}} \setminus \mathrm{D}(0,|c|))$ *to that of* z^2*, i.e.*

$$f_c(z) = \varphi_c^{-1}(\varphi_c(z)^2).$$

It satisfies $\varphi_c(z) = z + o(1)$ *at infinity. The function* $G_c(z) = \log|\varphi_c(z)|$ *is the Green function of the unbounded component of* $\mathcal{F}_c$*, with pole at* ∞*.*

In practice, the number of Newton steps necessary to jump from the level line λ_0 to a neighbourhood $\mathrm{D}(z,\varepsilon)$ of a root z is almost constant: on average, no more then 12 steps where necessary to split p_{41}, most of which can even be performed in a lower precision (see Table 3 in Section 6).

4.3 About multiple roots

As the level line is chosen above the highest modulus of critical values, the algorithm is not affected by multiple roots. A root of multiplicity m will be reached along m distinct iso-angles. In practice, our benchmarks (see Section 6.4.2) confirm that our algorithm is indeed robust to high multiplicities.

5 The certification process

As illustrated in [RSS17], there are various techniques to convince oneself that one has found all roots. For example, one can use Viete's identities to check that a few sums of the powers of the roots are in concordance with the first coefficients of the polynomial. Occasionally, this technique can provide some help in finding a few missing roots. However, for very large degrees, the cost of this checkup may become prohibitive for a result that, in the worst case, could be no more than a lucky coincidence.

Each of our results comes instead with a numerical proof, based on disk arithmetic. The complexity of checking that $P(z) = 0$ using our method is $O(d \log d)$.

5.1 How to prove the localization of a root

Let us consider $f \in \mathrm{Hol}(U)$ and assume that one has found a point $z_0 \in U$ such that $f(z_0) \in \mathrm{D}(0,\varepsilon)$ for some $\varepsilon > 0$ and $f'(z_0) \in \mathrm{D}(\lambda,\eta)$ with $|\lambda| > \eta > 0$. If $\varepsilon/(|\lambda| - \eta)$ is small enough, one can expect the existence of an exact root of f in the immediate vicinity of z_0. This idea is at the heart of Newton's method. We can combine it with the spirit of Rouché's theorem 11 to produce a quantifiable statement.

Theorem 6. *Given a holomorphic function* $f \in \mathrm{Hol}(U)$*, a disk* $B = \mathrm{D}(z_0, R)$ *such that* $\bar{B} \subset U$ *and* B' *a second disk such that* $f'(B) \subset B'$*, we assume that*

$$R\,\mathrm{dist}(0, B') > |f(z_0)|. \tag{37}$$

Then there exists a unique point $z_* \in B$ *such that* $f(z_*) = 0$*.*

Remark 8. *Note that, z_* is necessarily a simple root of f. In practice, disk arithmetic (see §5.3) provides an upper bound of $|f(z_0)|$ and, for a given $R > 0$, also of $|f'(z) - z_1|$ when $z \in B$, where z_1 is a numerical approximation of $f'(z_0)$. One can then construct B' and check if (37) is indeed satisfied.*

Proof. Let us introduce the arc $\gamma(t) = z_0 + Re^{2i\pi t}$ for $t \in [0,1]$. The fundamental theorem of calculus and the convexity of B' imply:

$$f(\gamma(t)) = f(z_0) + Re^{2i\pi t} \int_0^1 f'(z_0 + sRe^{2i\pi t})ds \in f(z_0) + Re^{2i\pi t} \cdot B'.$$

Assumption (37) thus implies that $f \circ \gamma$ is valued in an annulus centered at $f(z_0)$ that encircles zero. Its winding number with respect to zero is thus the same than that with respect to $f(z_0)$. Moreover, $f \circ \gamma$ is homotopic, within that annulus, to the path $t \mapsto f(z_0) + R\lambda e^{2i\pi t}$ where λ is the center of B' so the winding number is 1. Because of (37), $0 \notin f'(B)$ so all possible roots within B are simple and

$$|f^{-1}(0) \cap B| = \frac{1}{2i\pi} \int_\gamma \frac{f'(z)}{f(z)} dz = \frac{1}{2i\pi} \int_{f\circ\gamma} \frac{d\zeta}{\zeta} = \frac{1}{2i\pi} \int_{f\circ\gamma} \frac{d\zeta}{\zeta - f(z_0)} = 1$$

i.e. f admits exactly one root in B. □

5.2 How to prove the convergence of Newton refinements

The practical limitation of the classical estimates presented in Section 3.4 is that the radius on which Newton's method behaves bi-exponentially or exponentially is not, usually, an explicit one. The following result adresses this issue, at least for simple roots. For further results, see [Hen74, Chap. 6].

Theorem 7. *Let us consider a simple root z_* of $P \in \mathbb{C}[z]$ and $z_0 \in \mathrm{D}(z_*, \eta)$. For $\varepsilon > 3\eta$, if there exists a disk B' satisfying*

$$P'(\mathrm{D}(z_0, \varepsilon)) \subset B' \qquad \text{with} \qquad d(B', 0) > 2\,\mathrm{diam}(B'), \tag{38}$$

then one has $N_P(\mathrm{D}(z_0, \varepsilon)) \subsetneq \mathrm{D}(z_0, \varepsilon)$ and thus $\mathrm{D}(z_0, \varepsilon)$ is contained in the attraction bassin $\mathcal{A}(z_*) = \{z \in \mathbb{C}\,;\, \lim_{n\to\infty} N_P^n(z) = z_*\}$.

Remark 9. *The typical application case of Theorem 7 concerns an approximate root z_0 that is known, thanks to Theorem 6, to be in $\mathrm{D}(z_*, \varepsilon_1)$. To save resources, we want to publish z_0', which is an approximate value of z_0 with a reduced precision $\varepsilon_2 \gg \varepsilon_1$. Theorem 7 gives a condition that garanties that we can do so without losing any important information: an approximation of z_* to an arbitrary high precision can be retrieved from the sole knowledge of z_0' by applying iterates of N_P The assumption (38) can be checked using disk arithmetic (see §5.3). Note that, in the proof, the loss of $\frac{\varepsilon - 3\eta}{2}$ on the radius suggests that the Lipschitz constant of N_P is approximately Lipschitz of order $1 - \frac{3\eta}{2\varepsilon}$, which hints that the next iterations of N_P will converge at least exponentially.*

Proof. Let us consider $B = \mathrm{D}(z_0, \varepsilon)$ and $P'(B) \subset B'$ where $B' = \mathrm{D}(d, \varepsilon')$ is a disk such that $|d| > 5\varepsilon'$. In particular B' does not contain zero. Given $z_0 + h \in \mathrm{D}(z_0, \varepsilon)$, one has:

$$\begin{aligned} N_P(z_0+h) &= z_0 + h - \frac{P(z_0+h) - P(z_0) + P(z_0) - P(z_*)}{P'(z_0+h)} \\ &= z_0 + h\left(\int_0^1 1 - \frac{P'(z_0+th)}{P'(z_0+h)}dt\right) + (z_0 - z^*)\int_0^1 \frac{P'((1-t)z_* + tz_0)}{P'(z_0+h)}dt \end{aligned}$$

The ratio of two values $d + \vartheta_1$ and $d + \vartheta_2$ in B' satisfies:

$$\left|1 - \frac{d+\vartheta_1}{d+\vartheta_2}\right| = \frac{|\vartheta_2 - \vartheta_1|}{|d+\vartheta_2|} \leq \frac{2\varepsilon'}{|d| - \varepsilon'} < \frac{1}{2} \quad \text{and} \quad \left|\frac{d+\vartheta_1}{d+\vartheta_2}\right| \leq \frac{|d| + \varepsilon'}{|d| - \varepsilon'} < \frac{3}{2}.$$

One thus has $|N_P(z_0+h) - z_0| < \frac{1}{2}|h| + \frac{3}{2}|z_0 - z_*|$ *i.e.*

$$N_P(z_0+h) \in \mathrm{D}\left(z_0, \frac{\varepsilon}{2} + \frac{3\eta}{2}\right) = \mathrm{D}\left(z_0, \varepsilon - \frac{\varepsilon - 3\eta}{2}\right) \subsetneq \mathrm{D}(z_0, \varepsilon)$$

provided $\varepsilon > 3\eta$. As $\mathrm{D}(z_0, \varepsilon)$ is forward invariant under N_P, this disk is contained in its Fatou set. The iterates of N_P on $\mathrm{D}(z_0, \varepsilon)$ thus converge to z_*. □

5.3 Results on how to control numerical errors

Lastly, we need a theoretical background for handling all the errors that occur within the numerical computations that are necessary to prove that the Theorems 6 and 7 can indeed be applied. It is often (wrongly) believed that integer arithmetic is the only one apt for formal verification. We intent to show here that proofs can also be carried in floating point arithmetic.

The certification of a computation is only possible if the implementation choices respect a universal norm, as for example the one defined by the IEEE standards [IEE]. We use the library MPFR [MPFR] because it offers a reliable implementation of arbitrary precision that has been extensively tested. One of its main feature is the guaranty of proper handling of roundings, which is essential for the certification process described below. However, any other compliant implementation could equally be used as a foundation.

Working with complex numbers requires special care because, contrary to the real case, multiplications in $\mathbb{C}$ are not an elementary operation. In what follows, we will restrict our attention to the fields operations: $+$, $\times$.

5.3.1 Finite precision arithmetic

An ideal model of arithmetic with a finite precision $N \in \mathbb{N}^*$ consists in considering the following discrete subset of real numbers

$$\hat{\mathcal{R}}_N = \{0\} \cup \bigcup_{e \in \mathbb{Z}} 2^e \mathcal{Z}_N, \tag{39}$$

where

$$\mathcal{Z}_N = \left\{ \pm\Big(2^{-1} + \sum_{j=2}^{N} b_j 2^{-j}\Big) \text{ with } b_2, \dots, b_N \in \{0,1\} \right\} = \pm 2^{-N} [\![2^{N-1}, 2^N - 1]\!]. \quad (40)$$

When $x \in 2^e \mathcal{Z}_N$, one defines $e = e(x)$ as the *exponent* of x. The $(b_j)_{1,\dots,N}$ are called the *bits* of x, with the convention that $b_1 = 1$. Let us point out that if $N' \geq N$ is a larger precision, then $\hat{\mathcal{R}}_N \subset \hat{\mathcal{R}}_{N'}$.

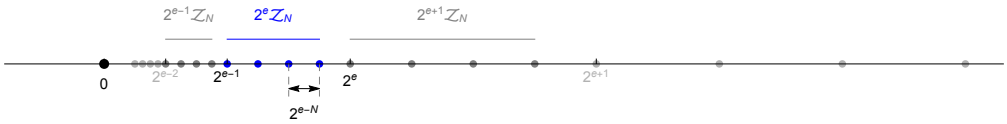


Figure 8: Representation of $2^e \mathcal{Z}_4 \cap \mathbb{R}_+$ for five consecutive values of e. Note how the gap between adjacent vertices varies with the exponent e, and the special role of zero as an accumulation point of $\hat{\mathcal{R}}_N$.

Practical implementations of finite precision arithmetic are restricted by physical contingencies; one then considers instead the finite set:

$$\mathcal{R}_N = \{\pm 0\} \cup \bigcup_{e=e_{\min}}^{e_{\max}} 2^e \mathcal{Z}_N \cup \{\pm\infty, \mathrm{NaN}\}, \quad (41)$$

where $-e_{\min}$ and $e_{\max}$ are (configurable but fixed) large positive integers. The additional symbols allow one to handle numericals exceptions without producing errors (NaN stands for *not a number*). For efficient processing by the hardware, bits are packed in groups of 8 called a *byte*, and packs of bytes (typically 4 or 8) are called *limbs*.

The arithmetic operations $+_N$ and $\times_N$ are naturally defined on $\hat{\mathcal{R}}_N$ by

$$\forall x, y \in \hat{\mathcal{R}}_N, \qquad x +_N y = \hat{\mathrm{R}}_N(x + y) \quad \text{and} \quad x \times_N y = \hat{\mathrm{R}}_N(x \times y),$$

where $\hat{\mathrm{R}}_N : \mathbb{R} \to \hat{\mathcal{R}}_N$ is the rounding operator that rounds to the nearest lattice point. Let us also define the operators $\hat{\mathrm{R}}_N^{\pm}$ that round respectively always up and always down, and the practical ones R_N, $\mathrm{R}_N^{\pm}$, that are valued in $\mathcal{R}_N$. By a convention, called *flush to zero*, which is not an IEEE standard but is the choice made in the MPFR library, $\mathrm{R}_N(x) = 0$ if $|x| < 2^{e_{\min}}$ even though 0 is not the nearest lattice point if $|x| > 2^{e_{\min}-1}$. Similarly, $|\mathrm{R}_N(x)| = \infty$ if $|x| > 2^{e_{\max}}(1 - 2^{-N})$.

For a given number $x \in \hat{\mathcal{R}}_N \backslash \{0\}$, the gap that separates it from its farthest immediate neighbors is

$$\widehat{\mathrm{ulp}}(x) = 2^{e(x)-N}. \quad (42)$$

By convention, $\widehat{\mathrm{ulp}}(0) = 0$. The name of this operator is *unit on last position* because it reflects the metric effect of a change of one unit on the least significant bit b_N. If

$x \in \mathcal{R}_N$ is a regular number *i.e.* if $e_{\min} \leq e(x) \leq e_{\max}$, one sets $\operatorname{ulp}(x) = \widehat{\operatorname{ulp}}(x)$. By convention, $\operatorname{ulp}(\pm 0) = 2^{1+e_{min}}$ and $\operatorname{ulp}(\pm\infty) = \infty$, which ensures the following statement. Note that the usual implementation choice is $e(0) = e_{\min} - 1$.

Proposition 5.1. *For $x \in \mathbb{R}$, one has*

$$\left|\hat{\mathrm{R}}_N(x) - x\right| \leq \tfrac{1}{2}\widehat{\operatorname{ulp}}(\hat{\mathrm{R}}_N(x)) \qquad \textit{and} \qquad |\mathrm{R}_N(x) - x| \leq \tfrac{1}{2}\operatorname{ulp}(\mathrm{R}_N(x)).$$

Remark 10. *IEEE standards require gradual underflow instead of flush to zero. This means that $\mathcal{R}_N$ should be complemented with a set of denormalized numbers $\{k2^{e_{min}-N}\,;\, k \in \mathbb{Z},\ |k| < 2^{N-1}\}$, called subnormals, which ensures that the lattice $\mathcal{R}_N$ is regular near zero. In that case, $\operatorname{ulp}(x) = 2^{e_{min}-N}$ for all those additional points, including $x = 0$. With this alternate convention, Proposition 5.1 still holds. The MPFR library offers the possibility of emulating the norm, but it is not the default behavior.*

5.3.2 Interval arithmetic

Interval arithmetic is a standard topic in numerical analysis [vdH09], [Rok01] whenever dependable results are critical. Reliable and fast libraries exist, like MPFI [RR05] or Arb [ARB].

Given a continuous numeric function f, the goal of interval arithmetic is to bound, as accurately as possible, the set to which $f(x)$ belongs when the prior knowledge on x is limited to a set of inequalities, *e.g.* $a \leq x \leq b$. The main challenge is to take dependency into account: for example, $f(x) = x^2 + x$ maps $[-1, 1]$ to $[-\frac{1}{4}, 2]$ while $g(x, y) = xy + x$ maps $[-1, 1]^2$ to $[-2, 2]$.

The following statement is an immediate but essential consequence of Proposition 5.1.

Proposition 5.2. *For $x_a, x_b \in \mathcal{R}_N$ and $r_a, r_b \in \mathcal{R}_M$, one has:*

$$\mathrm{I}(x_a, r_a) + \mathrm{I}(x_b, r_b) \subset \mathrm{I}\left(x_a +_N x_b\,,\, \mathrm{R}^+_M\left(r_a + r_b + \tfrac{1}{2}\operatorname{ulp}(x_a +_N x_b)\right)\right),$$
$$\mathrm{I}(x_a, r_a) \times \mathrm{I}(x_b, r_b) \subset \mathrm{I}\left(x_a \times_N x_b\,,\, \mathrm{R}^+_M\left(r_a r_b + r_a|x_b| + r_b|x_a| + \tfrac{1}{2}\operatorname{ulp}(x_a \times_N x_b)\right)\right)$$

where $\mathrm{I}(x, r) = (x - r, x + r)$ denotes open intervals along the real line.

In practice, the new radius is computed using the R^+_M operator for each intermediary computation to ensure that one gets a certifiable upper bound.

5.3.3 Naive rectangle arithmetic

For our purpose in complex dynamics, a naive use of a tensorized interval arithmetic faces the following shortcoming (see fig. 9): if a is in $z + [-r, r] + i[-r, +r]$ with $|z| = 1$, then a^2 belongs to $z^2 + (2r|z|_1 + r^2)\,[-1, 1] + i(2r|z|_1 + 2r^2)\,[-1, 1]$, with $|z|_1 = |\operatorname{Re} z| + |\operatorname{Im} z|$, which means that, when $\operatorname{Arg} z \simeq \pi/4$, the size of the uncertainty box is roughly multiplied by $2\sqrt{2}$ instead of the factor 2 imposed by the derivative. If this happens at many iterations of the square function, the resulting precision loss is catastrophic.

The average value of $\log|e^{i\theta}|_1$ on the unit circle is $\beta = 0.2365$. By the Birkhoff ergodic theorem, for a typical starting point a with $|a| = 1$, computing n iterates of the map $z \mapsto z^2$ starting at a induces a cumulative multiplicative loss of precision of about $e^{\beta n}$. For example, for $n = 200$, a loss of precision of order of 10^{20} should be expected.

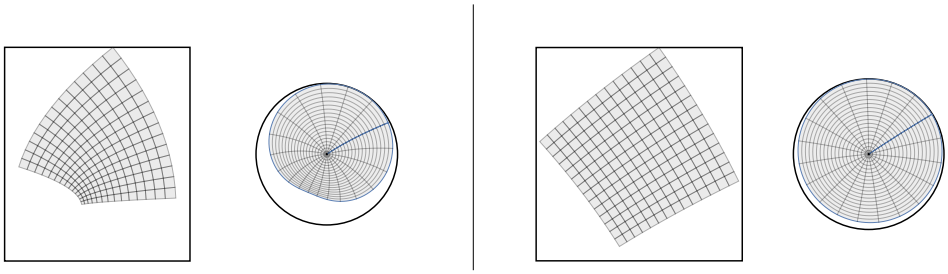


Figure 9: Comparison between the image by $f(z) = z^2$ of a square centered at $z = e^{3i\pi/16}$ of side $2r$, and that of a disks of center z and diameter $2r$ for $r = 0.5$ (left) and $r = 0.1$ (right). The images of the square and of the disk are drawn at the same scale, and compared respectively to an enclosing box or disk centered around z^2, which is of optimal size, provided that uniformity with respect to $\operatorname{Arg} z$ is required.

5.3.4 Disk arithmetic

The idea of disk arithmetic consists in studying what optimal outcome can be deduced from the prior knowledge that $a \in \mathrm{D}(z, r)$. While the idea is not new [PP98], it has remained, up to now, fairly uncommon.

The practical gain of substituting disks to squares is illustrated on fig. 9. If $a \in \mathrm{D}(z, r)$, then a^2 belongs to $\mathrm{D}(z^2, 2r|z|+r^2)$. If r is small enough to ensure that $r^2 \ll 2r|z|$ but without being necessarily tiny, the first observation is that each iteration of the square function multiplies the radius of the uncertainty disk by roughly a factor $2|z|$ instead of $2|z|_1$; one thus gains a casual $\sqrt{2}$ factor over the naive use of interval arithmetic.

The second observation is more profound and pertains to small values of r: for any conformal map f, the image of a small disk is almost a disk. Naturally, a nearly circular shape can be enclosed within a disk of a barely larger diameter. In practice, r is infinitesimally small and the derivative of the map on the disk of radius r is almost constant. In comparison to the unavoidable action of the differential $f'(z)dz$ at the center point, the additive loss on the bounding radius of $f(\mathrm{D}(z, r))$ is then of order r^2 and the multiplicative loss is thus about $1 + r/|f'(z)|$. For example, if we perform $k = 10^5$ iterations with bounded derivatives $|f'(z)| \geq 1$ along the orbit, starting from a radius $r = 10^{-20}$, then the cumulative multiplicative error of the disk arithmetic method is about $(1 + r)^k \simeq 1 + kr = 1 + 10^{-15} \simeq 1$.

Using disk arithmetic is the gateway to getting bounds that are both proven and almost optimal, even after a high number of iterates. It is a first essential step towards computational proofs in complex dynamics.

For $z_a = x_a + iy_a$, $z_b = x_b + iy_b \in \hat{\mathcal{R}}_N + i\hat{\mathcal{R}}_N$, the sum $z_a +_N z_b$ is naturally defined in components

$$z_a +_N z_b = (x_a +_N x_b) + i(y_a +_N y_b).$$

The radical difference between interval and disk arithmetic is that the computation of the product of complex numbers requires four exact multiplications on the real line, followed by two additions. Therefore, the product requires an intermediary precision $N' \geq N$:

$$z_a \times_{N,N'} z_b = (x_a \times_{N'} x_b) -_N (y_a \times_{N'} y_b) + i\left[(x_a \times_{N'} y_b) +_N (y_a \times_{N'} x_b)\right].$$

Intermediary products are guarantied exact only if $N' \geq 2N$. Let us extended the definition of ulp to complex finite precision numbers by

$$\mathrm{ulp}(x + iy) = \sqrt{\mathrm{ulp}(x)^2 + \mathrm{ulp}(y)^2}.$$

The following statement generalises Proposition 5.2 and is at the heart of the implementation of our library `[Mandel]`.

Theorem 8. *For centers $z_a, z_b \in \mathcal{R}_N + i\mathcal{R}_N$, radii $r_a, r_b \in \mathcal{R}_M$, one has:*

$$\mathrm{D}(z_a, r_a) + \mathrm{D}(z_b, r_b) \subset \mathrm{D}\left(z_a +_N z_b\,,\, \mathrm{R}^+_M\left(r_a + r_b + \tfrac{1}{2}\,\mathrm{ulp}(z_a +_N z_b)\right)\right).$$

For any intermediary precision $N' \geq N$, the product of exact centers satisfies:

$$z_a z_b \in \mathrm{D}\left(z_a \times_{N,N'} z_b, R_*\right),$$

where

$$R_* = \mathrm{R}^+_M\left(\tfrac{1}{2}\,\mathrm{ulp}(z_a \times_{N,N'} z_b) + \tfrac{1}{2}\sum_{\substack{u=x_a,y_a\\ v=x_b,y_p}} \mathrm{ulp}(u \times_{N'} v)\right).$$

The product of disks satisfies:

$$\mathrm{D}(z_a, r_a) \times \mathrm{D}(z_b, r_b) \subset \mathrm{D}\left(z_a \times_{N,N'} z_b\,,\, \mathrm{R}^+_M\left(r_a r_b + r_a|z_b| + r_b|z_a| + R_*\right)\right).$$

Finally, if $x \in \mathcal{R}_N$ and $r \in \mathcal{R}_M$, the scaling transform of the disk satisfies:

$$\mathrm{I}(x, r) \times \mathrm{D}(z_a, r_a) \subset \mathrm{D}\left(x \times_N z_a\,,\, \mathrm{R}^+_M\left((|x| + r)r_a + r|z_a| + \tfrac{1}{2}\,\mathrm{ulp}(x \times_N z_a)\right)\right).$$

The proof of the lemma is straightforward and the complex extension of the definition of ulp is illustraded on fig. 10. In our library, the role of R_* is implemented as an on-the-fly modification of ulp.

Remark 11. *In theory, there is a slightly tighter upper bound for the radius R_*, namely*

$$R^* = \mathrm{R}^+_M\Big\{\tfrac{1}{2}\,\mathrm{ulp}\Big(z_a \times_{N,N'} z_b + \mathrm{ulp}(x_a \times_{N'} x_b) + \mathrm{ulp}(y_a \times_{N'} y_b) + i\left[\mathrm{ulp}(y_a \times_{N'} x_b) + \mathrm{ulp}(x_a \times_{N'} y_b)\right]\Big)\Big\}.$$

However, the code complexity and computational cost of using R^* are unreasonably high compared to that of using R_* and are not justified for the expected gain.

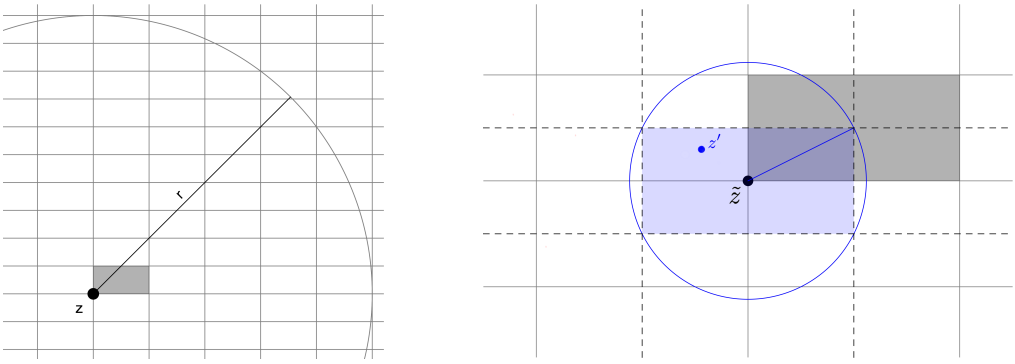


Figure 10: The key points to the proof of Theorem 8 : $\mathcal{R}_N \times \mathcal{R}_N$ grid (left) and the rounding strategy of $z' \in \mathbb{C}$ to $\tilde{z} \in \mathcal{R}_N \times \mathcal{R}_N$ within $\frac{1}{2}\operatorname{ulp}\tilde{z}$ (right).

To ease the tedious task of checking Theorem 8, let us recall a few ground rules. Having figure 10 in mind may help convey the key points. The center of a disk $\mathrm{D}_{z,r}$ is always considered exact; it belongs to $\mathcal{R}_N \times \mathcal{R}_N$. The corresponding ulp is thus a rectangle of dimensions $\operatorname{ulp} x \times \operatorname{ulp} y$ where $z = x + iy$. When computing the result of an operation, the new exact center $z' \in \mathbb{C}$ does not belong, in general, to the grid $\mathcal{R}_N \times \mathcal{R}_N$. The real and imaginary parts of z' are rounded to their closest value, which gives the new center $\tilde{z} \in \mathcal{R}_N \times \mathcal{R}_N$. To compensate, one needs to increase the radius of the disk by, at most, half the diagonal of the ulp rectangle. The claim that the center can be assumed exact is thus restored and one is ready for the next operation.

Remark 12. *The product of complex numbers is a multi-step operation over* $\mathbb{R}$*. As such, additional* ulp *have to be added to bound the successive rounding errors in each intermediary step.*

6 Implementation and numerical results

In this section, we present the library [Mandel] and the database [Mand.DB], which are a companion to this article and the numerical results that we have obtained with it. The main practical challenge is to preserve the efficiency of computations as the scale of the problem spans 12 orders of magnitude. The algorithm exposed in §4 is specifically designed for this. However, all the logistics regarding process scheduling and data handling have to keep up. At this scale, the certification of all the data becomes crucial, not only though the procedure described in §5 but also, at the lowest level, to ensure that no bit corruption occurs in the production and storage pipeline. In its final state, our database takes 43TB of disk space and we estimate (see Table 2) the overall cost to regenerate it from scratch to 83 years-core (723 000 hours-core) of computation time. The actual time that was actually necessary to build this library from scratch actually exceeds 1 million core-hours.

6.1 Appropriate data structures

Computations are performed with the standard `FP80` format (`long double`) for low precision or with `mpfr` and `mpc` numbers (a convenience wrapper for complex numbers) for arbitrary higher precisions. However, the foundation of scalable efficiency is the usage of appropriate data structures that are efficient for storage (both for disk and memory usage[7]), maintenance operations (sort, search, insertion) and that are easily compatible with high-precision arithmetic. In `[Mandel]`, we have developed such structures specifically for this project.

Our core numerical format to store complex numbers in $[-2,2)+i[0,4)$ is the `u128` format. It is composed of two unsigned 128-bits integers representing respectively the real and imaginary part with the two leading bits reserved for the mantissa. A pair (p,q) in `u128` format thus represents the complex number

$$z = (-2 + p \times 2^{-126}) + iq \times 2^{-126}. \tag{43}$$

Roots of p_n and $q_{\ell,n}$ are ultimately stored as `u128` numbers, which we call the *published value* in the rest of this text. The minimal absolute resolution of published values is therefore about 1.2×10^{-38}. This resolution is sufficient for our needs: for example, the minimal distance between two distinct hyperbolic centers of period 41 is only 2.45×10^{-23} so pairs of roots will differ by at least 50 bits.

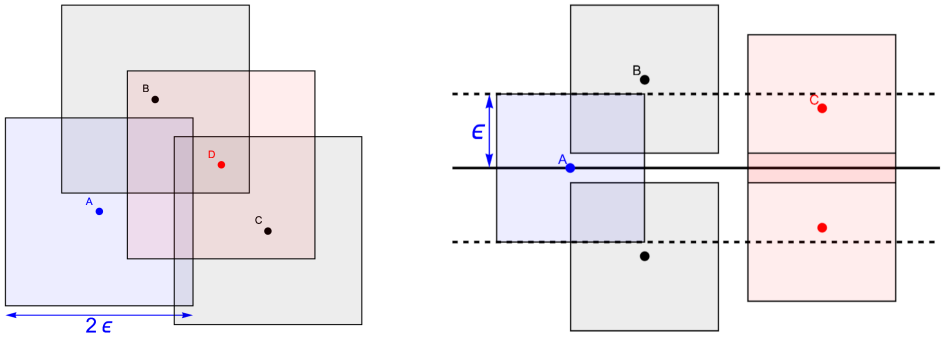


Figure 11: Insertion in `nset` is non commutative (left): inserting A, D, B, C results in $\{A, D\}$ while inserting A, B, C, D results in $\{A, B, C\}$. Near the real axis (right), A and B respect the assumption (44). However, C does not and could thus not be added to an `nset`.

The work-horse of our database is the `nset` structure. This structure implements the mathematical idea of a finite set of points in the rectangle $[-2,2)+i(-4,4)$ that is symmetric with respect to the real axis. Only the points in the upper-plane are stored. Each stored point in the set is a `u128` complex number. Each set comes with a separation

[7] Because of memory alignement, the size of a structure in RAM may not match its size on the disk.

parameter $\varepsilon > 0$ and all operations on the `nset` guaranty the following property: given two distincts points z_1, z_2 in the set or in its conjugate image:

$$\max\{|\operatorname{Re}(z_1 - z_2)|, |\operatorname{Im}(z_1 - z_2)|\} \geq \varepsilon. \tag{44}$$

In other words, points are considered as equal if one center belongs to the square "pixel" of side 2ε centered in the other one. Note that this is not an transitive relation (see Figure 11). Special care must be taken near the real axis to avoid collisions with the conjugate images ($z_2 = \overline{z_1}$). In our database, we use a separation $\varepsilon_h = 3.23 \times 10^{-27}$ for hyperbolic centers and $\varepsilon_m = 8.08 \times 10^{-28}$ for Misiurewicz points. Note that this parameter serves a different purpose than the certification parameters (see Fig. 12).

The points in an `nset` structure are totally ordered by the lexicographical order:

$$z_1 \preceq z_2 \qquad \Longleftrightarrow \qquad \operatorname{Re} z_1 < \operatorname{Re} z_2 \quad \text{or} \quad \begin{cases} \operatorname{Re} z_1 = \operatorname{Re} z_2, \\ \operatorname{Im} z_1 < \operatorname{Im} z_2. \end{cases} \tag{45}$$

To optimize memory management, an `nset` can exist in two internal states, either locked or unlocked. In its unlocked state, an `nset` contains a family of sub-sets (called bars) whose sizes form, ideally, a geometric progression. Each bar is ordered with (45). Bars are independent from one another (no ordering). Insertion is performed on the last (smallest) bar unless it is full, in which case the bar is merged with the previous one (recursively if necessary). This means that insertion can be performed in $O(1)$ to $O(N)$ operations, where N is the size of the `nset`. However, large numbers of operations are exponentially rare: the k^{th} bar is only merged into the bar $k+1$ when all the previous bars are full, so once in $O(2^k)$ insertions. If we denote by B is the size of the smallest bar and $K = \log_2\left(1 + \frac{N}{B}\right)$ the total number of bars, then, on average, an insertion costs

$$\frac{\displaystyle\sum_{k=1}^{K} kB2^{k-1}}{\displaystyle\sum_{k=1}^{K} B2^{k-1}} = \frac{1 + (K-1)2^K}{2^K - 1} = O(\log N)$$

memory management operations. Conversely, a search on an unlocked `nset` takes $O(\log^2 N)$ operations because each of the $K = O(\log N)$ bars must be searched independently. Note that in order to ensure (44), an insertion can only be performed after a search for the new point has confirmed that it is indeed appropriate to add it. Points near the real line (like the point C on the right-hand side of Figure 11) must be treated as exceptions: it is up to the user to decide wether it is appropriate to correct the imaginary part to zero or to refuse to add the point.

In its locked state, all the memory bars of an `nset` are merged, which means that the set becomes fully ordered. This is a one-time $O(N)$ cost. Insertion within a locked `nset` is not authorized anymore. Note that unlocking remains possible and, in that case, the previous logarithmic costs can be restored (using dyadically smaller bars before the

main data chunk). In a locked `nset`, the cost of a search drops to $O(\log N)$. A locked `nset` is essentially ready to be written to the disk. Actions on an `nset` structure require only a $O(\log N)$ memory overhead (pointers to the each memory bar), which turns out to be constant in practice[8]. Not that we haven't implemented deletion as a guaranty that no point will ever be added unless we know for certain that it should.

We have also implemented `pset`, *i.e.* planar sets, which is a structure analogous to the `nset` but based on `FP80` numbers (`long double`) instead of `u128`. The `pset` structure implements the mathematical idea of a finite set of points in the complex plane, with no geometric restrictions. It provides logarithmic search and insertion times with logarithmic memory overhead. The theoretical optimum [FS89], [Lar13] for operations on ordered memory structures is $O(\log N/\log\log N)$. The performances of our structures are therefore quite satisfactory and may be of general interest. In particular, aside from improved RAM usage, our low memory footprint minimizes the cost of the conversions to and from a disk-storage format, which is crucial when handling terabytes of data[9].

Our format to handle vectors of high-precision numbers is called `mpv`. This format is optimized for storage and large scale disk access. In a given vector, all the elements share a common precision. The vector can either be real or complex, with consecutive pairs of real/imaginary parts. Individual operations on elements are possible, though awkward. One important feature of this format is the possibility to concatenate multiple vectors into a single file (what we call mini-files) and to perform parallel read-writes on each mini-file.

Our implementation also contains tools to import, export and compare `CSV` files (comma separated values) of complex numbers of arbitrary precision (using a, b to represent $a+ib$). This format is extremely slow because of the binary to decimal conversion and requires about 1.8 times more disk space. It ensures however a human-readable output and a minimal compatibility with other computing or graphical tools.

6.2 Certification results and protection against data corruption

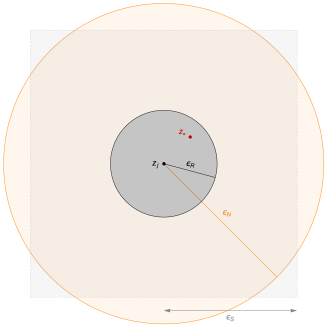
Each published value (see (43) for a definition) for a root of p_n or $q_{\ell,n}$ comes with multiple levels certifications. Let us underline that the following theorem contains about 3×10^{12} individual statements, whose proof are computer assisted.

Theorem 9. *There are constants ε_R, ε_N and ε_S given in Figure 12 for which the following holds. In the `nset` database for the roots of p_n or $q_{\ell,n}$, each published value $z_j \in 2^{-126}(\mathbb{Z}+i\mathbb{N})$ can be paired with a unique actual root $z_* \in \mathrm{D}(z_j,\varepsilon_R)$. This pairing is bijective in the upper half plane $\operatorname{Im} z \geq 0$. Each pair of published values z_j, z_k satisfies $|z_j - z_k| \geq \varepsilon_S$ and either $z_j \in \mathbb{R}$ or $|z_j - \bar{z}_j| \geq \varepsilon_S$. Lastly, the disk $\mathrm{D}(z_j,\varepsilon_N)$ is*

[8] No more than 64 memory bars are ever necessary if one uses `unsigned long` for array indices.

[9] A pointer requires typically 8 bytes on a 64-bit architecture; implementing *e.g.* a balanced tree of `FP80` with 2 pointers to the children would therefore induce about 50% of memory overhead.

entirely contained in the attraction bassin of z_* for the Newton method of the associated polynomial.



	$p_{n\leq 40}$	p_{41}	$q_{\ell,n}$
ε_R	10^{-30}		10^{-35}
ε_N	10^{-24}	10^{-25}	10^{-32}
ε_S	3.23×10^{-27}		8.08×10^{-28}

Figure 12: For each root z_* with $\operatorname{Im} z_* \geq 0$, the published value $z_j \in 2^{-126}(\mathbb{Z} + i\mathbb{N})$ comes with different radii. The radius of separation ε_S guaranties proper counting, the radius of certification ε_R is such that $z_* \in \mathrm{D}(z_j, \varepsilon_R)$ and the radius of convergence ε_N ensures that the $\mathrm{D}(z_j, \varepsilon_N)$ is contained in the bassin of attraction of z_* for the Newton method.

Proof. Using the controlled rounding features of the MPFR library [MPFR], we have implemented disk arithmetic *i.e.* Theorem 8. This allows us to provide, for each root candidate z_j, a numerical proof that the assumptions of Theorem 6 on the localisation of each root can indeed be applied. Thus, we can certify that the published coordinates of each root differ from an actual root by no more than 10^{-30} for p_n or 10^{-35} for $q_{\ell,n}$. Similarly, using disk arithmetic with ε_N, we can check the assumptions of Theorem 7 which ensures the claim on the Newton bassin. Next, each pair of two published values or any pair with conjugate values satisfies the separation assumption (44) because it is build into our storage structure (see §6.1). It is slightly stronger than the one claimed here. This separation guaranties that it is possible to count unambiguously all the real roots, and all the non real roots in the upper half-plane. Comparing with the theoretical count given by Theorem 1 and (12) ensures our bijection claim. □

Remark 13. *We are confident that each component of the published values is exact up to*

$$\pm\tfrac{1}{2}\operatorname{ulp}(\operatorname{Re}\texttt{u128}) = \pm 2^{-127} \simeq 5.9 \times 10^{-39},$$

which is better than the ε_R radius claimed in Theorem 9. Indeed, once we have a z_j value that passes the certifications for both disk arithmetic and Newton bassin, we can refine its value using Newton's method until reaching a fixed point at the desired precision (43), up to a final rounding error.

At the terabyte scale, the possibility of bit corruption becomes a significant issue. Besides the previous statement, practical precautions must therefore be taken to ensure the integrity of the data along the whole production chain, from the initial computation

that finds a root to its final storage in a file. This protection must also extends to any subsequent use of the stored values.

Our library [Mandel] incorporates a strict data certification procedure. Our `nset` file format implements a header that contains the MD5 checksum of the data stored in the rest of the file. When writing a file, a checksum of the original data is first computed in memory, then the file is written onto the disk and the checksum is written in the file header. To detect subsequent data corruption (due to an error in a file transfert or a random bit flip), each time a file is loaded, the checksum of the data stored in the file is computed again and checked against its original value stored in the header. This protocol ensures the data integrity once the original MD5 stamp has been generated.

Lastly, one needs to check that a random memory corruption has not affected the initial creation process, after the value was computed but before the original MD5 checksum was generated. The only sensible way to guard against this problem is to perform a new independent certification (count and proofs) of Theorem 9 of all the data written on the disk, which we did using our applications `hypCount`, `hypProve`, `misCount` and `misProve`. To encourage independent verifications, the code of those applications has been kept as minimalistic as possible.

The only uncertainty left is a data corruption that could happen after this second certification but would remain undetected by the MD5 checksums. As MD5 is a 128 bit cryptographic hash function[10], the probability of such an event is of order $2^{-128} \simeq 3 \times 10^{-39}$.

6.3 Quick single-core splitting of polynomials up to degree 10^9

Splitting the polynomials p_n and $q_{\ell,n}$ for the first values of n is a standard computational challenge. Using the standard algorithms of Section 3, splitting p_{25} (degree 1.7 million) is a task that used to lie at the computational frontier and required days of core usage. Using our new algorithm based on level lines (presented in Section 4), the polynomial p_{25} can now be split in only a few minutes on a single core of consumer hardware. As illustrated of Figure 13, we push the day-core limit back to p_{33} (degree 4.3 billion). As explained in §6.3.2 below, this degree is the highest one that can be handled using hardware arithmetic.

6.3.1 Implementation of the level line algorithm

In our implementation [Mandel], the application `hypQuick` splits p_n and computes a listing of $\mathrm{Hyp}(n)$. On Figure 13, the low degree scatter for processes that execute in less than 5 seconds is driven by memory loading and is not significant. The 40% jump in the average processing time per root at p_{30} is due to the fact that we double the number of points on the initial level line when $n \geq 30$. The code of this application is concise and is largely independent of the rest of the library. It relies on the `pset` structure (see

[10] Granted that our database is not subject to a malicious attack that would actively seek a vulnerability.

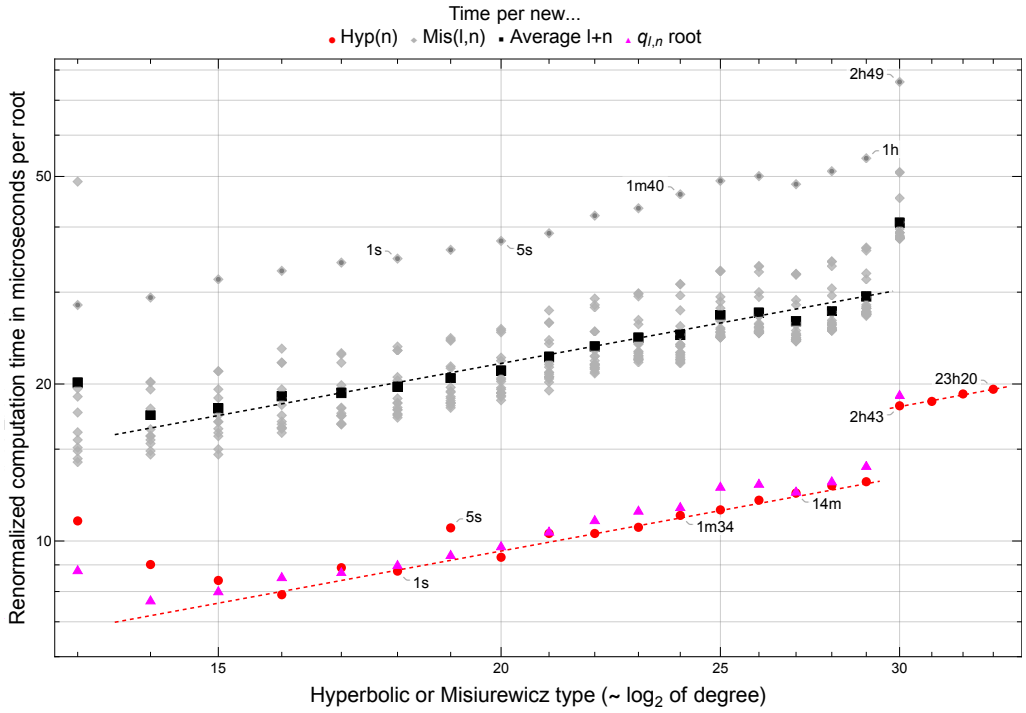


Figure 13: Computation time, in microseconds, per new parameter Hyp(n) computed with `hypQuick` (red dots) or Mis(ℓ, n) with `misQuick` (gray lozenges). The black square give the average time at a given $\ell + n$. The slowest subtype corresponds to $q_{\ell,2}$. Labels indicate the total splitting time of p_n or $q_{\ell,2}$ on a single CPU. The magenta triangles represent the average computation time per root of $q_{\ell,n}$, regardless of wether the root is a hyperbolic divisor or not. The dashed lines correspond to a slope $O(n^{0.8})$ or $O((\ell+n)^{0.8})$, and the overall fit suggests that the algorithm behaves, in practice, as $O(d(\log d)^{0.8})$.

§6.1) to collect and sort the roots and prevent duplicates. The functions that implement the Newton's method are common to our whole library.

Let us comment on our implementation for the splitting of q_n. The endpoints of the level curve $|p_n(z)| = 5$ along the real line are found by dichotomy. The discrete level curve $\mathbb{L}_{5,2^{n-1}}(p_n)$ defined by (31) is computed on the fly using Proposition 4.1 with $M = 8$ points per turn if $n \leq 29$ or $M = 16$ points per turn if $n \geq 30$, *i.e.* one computes solutions of $p_n(z_k) = 5e^{2ik\pi/M}$, each z_k acting as the starting point for the Newton iterations towards z_{k+1} (see Figure 6). For each z_k such that $k \equiv 0 \mod 2$ (if $n \leq 29$) or $k \equiv 0 \mod 4$ (if $n \geq 30$), a Newton descend attempts to find a new root of p_n. To prevent divergent trajectories, the number of iterations is capped by $O(\log d)$. The new root is then searched in $O(\log d)$ within a `pset` structure that contains all the roots of the known divisors of p_n given by Theorem 1. If the root does not belong to a divisor, it is added, in constant time, into a `pset` structure that collects all new roots. Then the computation of the discrete level curve restarts and the process continues. Overall, the

complexity of splitting p_n and identifying $\mathrm{Hyp}(n)$ using this implementation is $O(d \log d)$ with $d = 2^{n-1}$, which explains the record times shown on Figure 13. All computations are performed using `FP80` hardware arithmetic. The memory requirement is $d + O(\sqrt{d})$ to store both the divisors and the new roots. The application finally outputs the listing of the set $\mathrm{Hyp}(n)$ in `CSV` format.

We have also implemented two applications that split $q_{\ell,n}$ for $\ell+n \leq 30$. The application `misQuick` uses the original polynomials $q_{\ell,n}$, which have high-multiplicity roots, as described in Theorem 2. Figure 13 synthesizes our benchmarks. The computation time appears to obey a general rule $O(d(\log d)^{0.8})$ per new parameter, regardless of multiplicity. The constant factor in front of this experimental law depends on the sub-family of polynomials on which the benchmark is run and the parameters (like the ratio of starting points per root). In theory, a $O(d \log d)$ scaling law should have been observed. Let us point out that the renormalized computation time per root (regardless of wether it is a Misiurewicz type or not) is equivalent to that of hyperbolic polynomials of the same degree. This means that high-multiplicities have, at least in that case, a minimal effect on the efficiency of our algorithm. To get rid of multiplicities, the application `misSimpleQuick` uses the simplified polynomials $s_{\ell,n} = p_{\ell+n-1} + p_{\ell-1}$ instead of $q_{\ell,n}$. This polynomial (9) plays a central role in the proof of Theorem 2 (see equation (59) in Appendix A), it has simple roots that contain $\mathrm{Mis}(\ell, n)$ and is a lot simpler to compute than the fully reduced polynomial $m_{\ell,n}$ given by (11).

6.3.2 Reaching the limit of hardware arithmetic

The different `C` standard types of finite precision arithmetic are recalled in Table 1. For example, `FP80` arithmetic is the highest precision available on common hardware and offers 64 significand bits; as such, it is suitable to represent the list of all roots of a polynomial only when the minimal root separation exceeds $2^{-64} \simeq 5 \times 10^{-20}$. Moreover, a margin of at least a few bits is necessary for the Newton dynamics to be meaningful and converge to those roots.

Norm	Standard C	Java	Sign	Exponent	Significand	RAM	Disk
FP32	`float`	`float`	1 bit	8 bit	1+23 bit	4B	4B
FP64	`double`	`double`	1 bit	11 bit	1+52 bit	8B	8B
FP80	`long double`	-	1 bit	15 bit	64 bit	12-16B	10B
FP128		-	1 bit	15 bit	1+112 bit	16B	16B

Table 1: Standard types of finite precision arithmetic.

Let us consider the two left-most real roots $c_1 = -2 + \varepsilon_1$ and $c_2 = -2 + \varepsilon_2$ of p_n, with $0 < \varepsilon_1 < \varepsilon_2$. For large n, the corresponding dynamics (see Fig. 14) linger near 2 for most of the trajectory and separate from one another only at the last step, with

$$p_{n-1}(c_1) = \sqrt{2-\varepsilon_1} \simeq \sqrt{2}\left(1 - \frac{\varepsilon_1}{4}\right) \quad \text{and} \quad p_{n-1}(c_2) = -\sqrt{2-\varepsilon_2} \simeq -\sqrt{2}\left(1 - \frac{\varepsilon_2}{4}\right),$$

hence the rough estimate:

$$2\sqrt{2} \simeq |p_{n-1}(c_1) - p_{n-1}(c_2)| \simeq |p'_{n-1}(-2)||c_1 - c_2|. \tag{46}$$

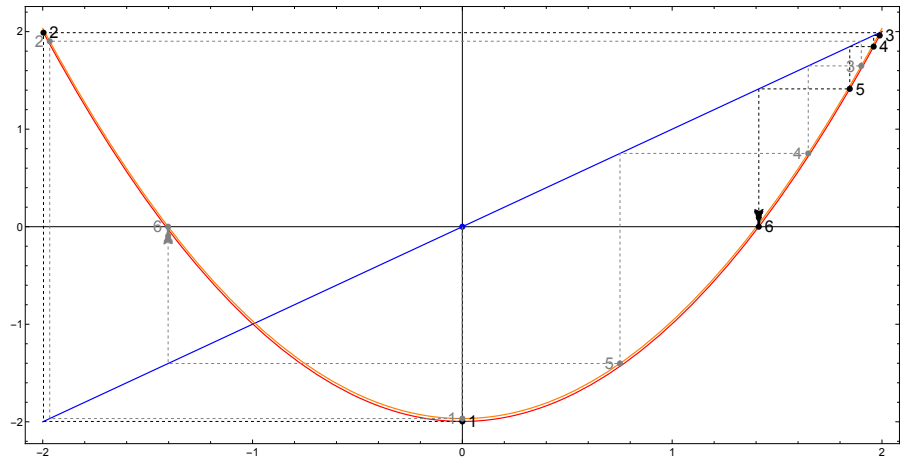


Figure 14: Dynamics associated with the left-most real roots of p_n (here with $n = 6$).

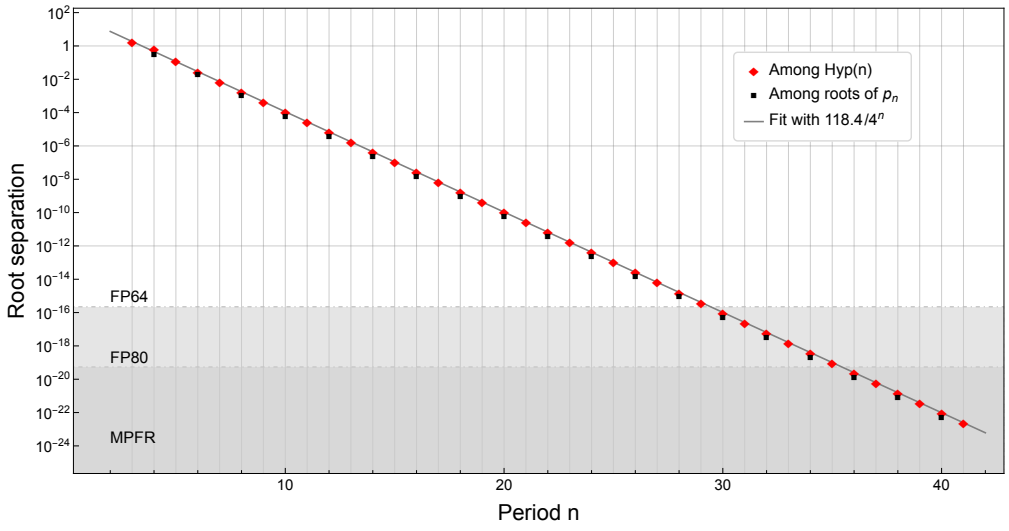


Figure 15: Minimal distance among hyperbolic centers Hyp(n) and among the roots of p_n (which differs only for odd n). The shaded regions highlight the resolution of hardware finite precision arithmetics `FP64` and `FP80` and the region where a higher precision is required.

The computation of the derivative is classical and is based on the connection between

the parameter space and that of dynamics. Indeed, as $f_c'(z) = 2z$ and

$$(f_c^k)'(z) = \prod_{\ell=0}^{k-1} f_c'(f_c^\ell(z)) = 2^k z f_c(z) \dots f_c^{k-1}(z),$$

the recursive definition of p_n gives

$$p_n'(c) = 1 + \sum_{\ell=1}^{n-1} (f_c^\ell)'(p_{n-\ell}(c)) = 1 + \sum_{\ell=1}^{n-1} 2^\ell p_{n-\ell}(c) p_{n-\ell+1}(c) \dots p_{n-1}(c)$$

thus

$$\frac{p_n'(c)}{(f_c^{n-1})'(c)} = 1 + \sum_{\ell=0}^{n-2} \frac{2^\ell p_{n-\ell}(c) p_{n-\ell+1}(c) \dots p_{n-1}(c)}{2^{n-1} p_1(c) p_2(c) \dots p_{n-1}(c)} = 1 + \sum_{k=1}^{n-1} \frac{1}{(f_c^k)'(c)} \tag{47}$$

For $c = -2$, one has $f_{-2}^k(-2) = 2$ for all $k \geq 1$ and $(f_{-2}^k)'(-2) = -4^k$. Substitution in (47) gives $p_n'(-2) = -\frac{4^n+2}{6}$ and thus, within the approximation (46):

$$|c_1 - c_2| \propto 4^{-n}. \tag{48}$$

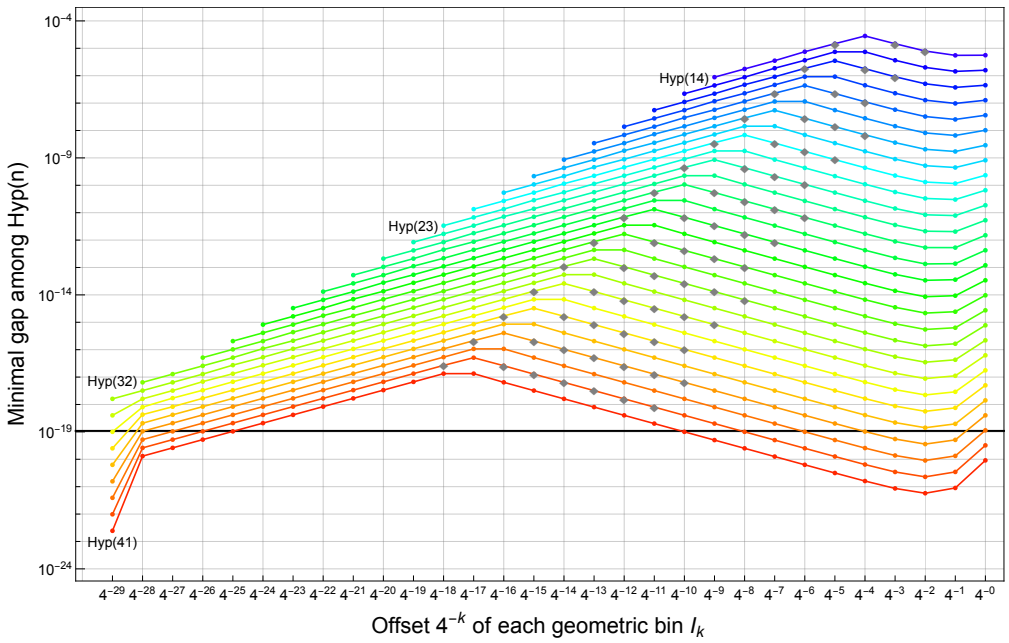


Figure 16: Histogram of the minimal distance among $\mathrm{Hyp}(n) \cap I_k$ with geometric bins $I_k = (-2+4^{-k-1}, -2+4^{-k}] \times \mathbb{R}$ illustrating that the closest gap happens near $c = -2$. The gray data points corresponds to bins where the minimal distance among all roots of p_n is strictly smaller (close proximity of a divisor).

The actual root separation computed on our high-resolution dataset is illustrated on Figures 15, 16 and 17. Within $\mathrm{Hyp}(n)$, one can check numerically that the two left-most real roots are, indeed, the closest points. The experimental scaling law matches the predicted behavior (relative error smaller than 0.1% when $n \geq 9$):

$$\min_{\substack{c,c' \in \mathrm{Hyp}(n) \\ c \neq c'}} |c - c'| \sim \frac{118.4}{4^n}. \tag{49}$$

For odd n, both roots lie at the center of a primitive component of period n. For even n, the minimal distance among roots of p_n is slightly smaller than among $\mathrm{Hyp}(n)$, due to the presence of a closer pair involving a root of a divisor (see Figure 16): one root is the center of a primitive component of period $n/2$ and the other one is the center of a satellite of period n.

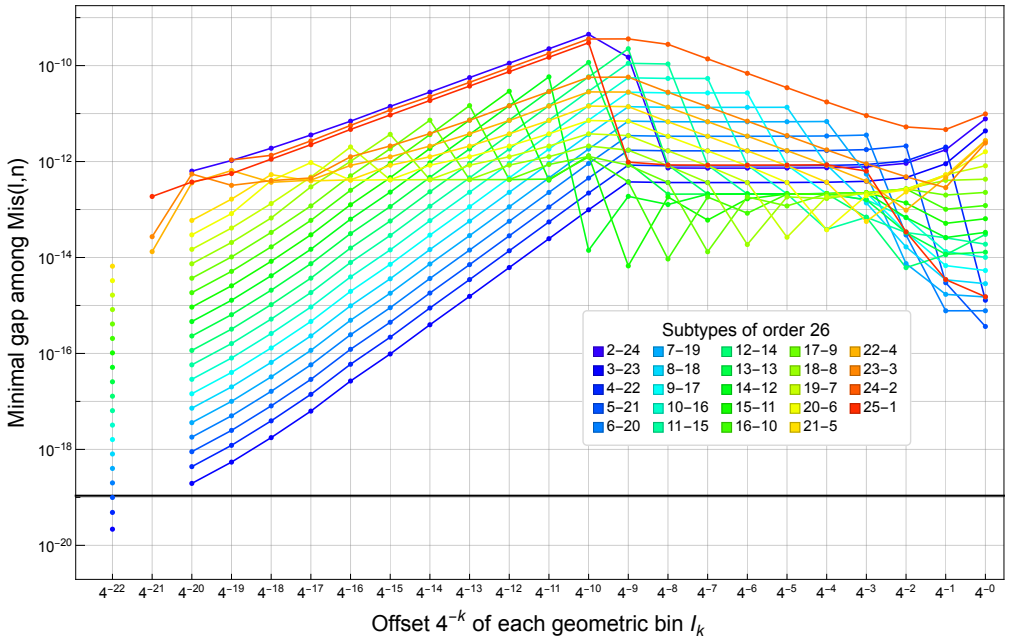


Figure 17: Histogram of the minimal distance among $\mathrm{Mis}(\ell, n) \cap I_k$ for $\ell + n = 26$, which is the first Misiurewicz order that cannot be integrally represented with `FP80` arithmetic. The separation within $\mathrm{Mis}(3, 23)$ and $\mathrm{Mis}(4, 22)$ drops below the limit of the representation.

For $\mathrm{Mis}(\ell, n)$, the limitation of hardware arithmetic is first hit for the subtypes $\mathrm{Mis}(3, 23)$ and $\mathrm{Mis}(4, 22)$, as illustrated on Figure 17. Both sets contain, at the leftmost end of $\mathcal{M}$, two distincts real parameters separated by less than $4.86 \times 10^{-20} < 2^{-64}$. They cannot be separated in `FP80` arithmetic. More generally, we have observed that all sets $\mathrm{Mis}(\ell, n)$ with $26 \leq \ell + n \leq 32$ and $3 \leq \ell \leq 3(\ell + n - 25) + 1$ contain, near the left tip, parameters that cannot be represented in `FP80` arithmetic. Obviously, our app

`misQuick` is subject to those limitations and reports the missing roots, which is normal behavior.

However, to offer a user friendly interface, the apps `misQuick` and `misSimpleQuick` provide an option to refine all roots near the real axis and left of $-2+4^{-16}$ using high-precision `MPFR` numbers. This option does not sacrifice much of the performances and allows us to compute a complete list of all $\mathrm{Mis}(\ell,n)$ parameters of type $\ell+n\leq 30$ in the time-frame illustrated on Figure 13.

When $\ell+n=30$, a second region (near $\mathrm{Re}\,c=-1$) contains close complex pairs (see Figure 21) that are separated by less than 2^{-62}. Computing types of order $\ell+n\geq 31$ would require introducing multiple high-precision exceptions. At that point, one should just switch to our HPC implementation described in Section 6.4.

Those numerical observations establish that, in the family of all hyperbolic centers and Misiurewicz parameters, the roots of p_{33} and $m_{\ell,n}$ with $\ell+n=30$ constitute the final frontier of what can be computed using `FP80` hardware arithmetic. As illustrated on Figure 13, our new algorithm based on level-lines can reach this frontier in just a bit less than one day-core for the hyperbolic parameters and in about 3 hours for each pre-periodic type (half that for `misSimpleQuick`).

Remark 14. *The upper exponent limit of `FP80` numbers is* $2^{2^{14}}\simeq 1.18\times 10^{4932}$. *In `hypQuick`, we deal with polynomials of degree up to* 2^{32}. *Let us underline that*

$$|z|^{2^{32}}\leq 1.18\times 10^{4932}\quad\Leftrightarrow\quad |z|\leq R_{\texttt{FP80}}\quad \textit{with}\quad R_{\texttt{FP80}}\simeq 1+2.64\times 10^{-6}$$

and that $\mathcal{M}\cap \mathrm{D}(0,R_{\texttt{FP80}})^c\neq\emptyset$. *To compute accurately one Newton step* $p_n(z)/p_n'(z)$ *when we detect that both the numerator and denominator are so large that each will exceed the exponent limit (which happens in the first steps), we simplify a common large factor, without loss of precision.*

6.3.3 A-posteriori certification of the `FP80` results

There is no realistic hope of certifying `FP80` computations within their own framework. The operations on `long double` are extremely hardware dependent and we have no real garanties that a given implementation will perform properly, in any circumstances, up to the last bit. Implementing disk arithmetic is possible and can provide a reasonable amount of confidence and is therefore included in our library. However, using disks based on hardware arithmetic still leaves an unacceptable room for doubt[11] that only `MPFR` (which implements guarantied rounding directives) can waive.

For all periods 3 to 33, we did compare the listing of $\mathrm{Hyp}(n)$ obtained in `FP80` arithmetic using `hypQuick` to the certified listing obtained independently with high-precision `MPFR` and certified disk arithmetic (see §5 and §6.2). The maximum deviation between two corresponding lists does not exceed 5.24×10^{-19} (reached, in our case,

[11] A single disk radius mistakenly rounded the wrong way breaks the logic chain of who contains whom.

for p_{24}). In particular, the lists of roots are identical up to a precision of 10^{-18}. This a-posteriori check constitutes the best possible certification of our `FP80` results.

Similarly, the a-posteriori certification of $\mathrm{Mis}(\ell, n)$ for $\ell + n \leq 25$ reports a maximum deviation of 3.25×10^{-19} between the lists obtained in hardware arithmetic and those obtained with `MPFR`. Subtypes of higher order also pass the certification, with an obvious exception for the parameters that cannot be represented in hardware arithmetic and are thus out of the reach of `misQuick` (see Section 6.3.2). Of course, they can be certified only if one activates the high-precision option for parameters left of $-2 + 4^{-16}$.

6.4 Our HPC approach for splitting a tera-polynomial

Scaling up our algorithm from the splitting of p_{33} to that of p_{41} requires the resources from a HPC center. The two key ingredients are massively parallel operations and the ability to switch on the fly between hardware arithmetic, and arithmetic in arbitrary precision. The splitting process is decomposed in multiple stages.

6.4.1 Overview of the splitting process

The first stage, called is a massively parallel raw search that involves $J = 2^{n-27}$ tasks (respectively $J = 2^{n+\ell-27}$ for $q_{n,\ell}$) that are independent of each other. Each task consists in finding roots in a certain sub-region of the Mandelbrot set. More precisely, the computation of the discrete level line $\mathbb{L}_{5,2^{n-1}}(p_n)$ is split in J pieces of equal cardinality. We choose $2^{n-1}/J = 2^{27}$ to ensure that the resulting `nset` files of roots would take about 1.1GB each on disk. Each task computes a part of the finest level line and finds all the roots that can be reached from it using the Newton map. In the library, the corresponding apps are `hypRaw` and `misRaw`. The alternative `misSimpleRaw` uses the simplified polynomials $s_{\ell,n} = p_{\ell+n-1} + p_{\ell-1}$ defined by (9) instead of $q_{\ell,n}$, which eliminates most hyperbolic divisors and ensures that all roots are simple.

The superiority of our level-line algorithm is illustrated on Figures 7 and 18: sections of equal cardinality of $\mathbb{L}_{\lambda,N}$ span physical regions of extremely varied size. The level-line is a naturally self-refining mesh in the sense that the mesh is, by construction, denser in regions where $\arg P(z)$ cycles more rapidly, which indicates an intrication of the bassins of attraction for the Newton flow and is thus likely to occur for the Newton map too. As iso-angles (24) and external rays coincide asymptotically at infinity, it can be expected that those regions still corellate with a higher density of external rays, and thus capture faithfully, as the ray land, the harmonic measure of $\mathcal{M}^c$ and thus the roots of p_n and $q_{\ell,n}$ (see Figure 20 and Section 2.3).

The key to the parallelization is the possibility to compute recursively a point with a given phase on a level line. One uses discrete iso-angle trajectories (25), which are good approximations of the external rays (Figure 19). A point on $\mathbb{L}_{\lambda,N}(p_n)$ with a given phase can be obtained from a corresponding point in $\mathbb{L}_{\lambda',N/2}(p_{n-1})$ with $\lambda' > \lambda$ using only a few Newton steps. If λ' is large enough, one can be confident that no skip modulo 2π has occurred so the phase is correct. This process is explained in details in Section 4.2.

Use the app `levelSets` to initiate the low-resolution level sets that will be used by each parallel task.

The second stage consist in merging the roots found by all the parallel tasks. Sorting tera-bytes of data split in thousands of files of about one gigabyte and deleting the roots found multiple times appears, at first, as a substantial computational challenge. However, as explained in Section 6.1, we used an appropriate, custom made, data structure that allows for logarithmic search and insertion times and prohibits duplicates. Overall, the time requirements for the second stage turned out to be negligible (see Table 2). The corresponding apps are `hypRawCount` and `misRawCount`. They can also generate `maps`, which are low-resolution portraits of the Mandelbrot set that count how many roots of a given type end up in a given pixel (see Figure 20).

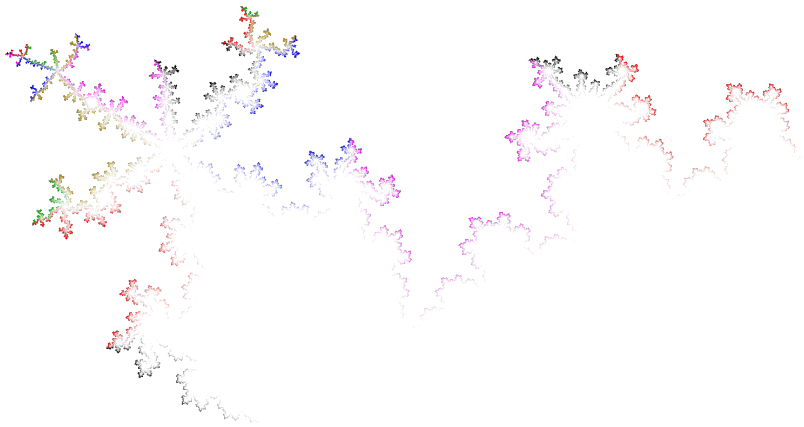


Figure 18: Zoom on 18 consecutive raw jobs for p_{35} : the roots found by each parallel task are colored differently from the next one. Note the extreme spatial variability between tasks, even though each one starts from a piece of the discrete level line that contains 2^{27} points.

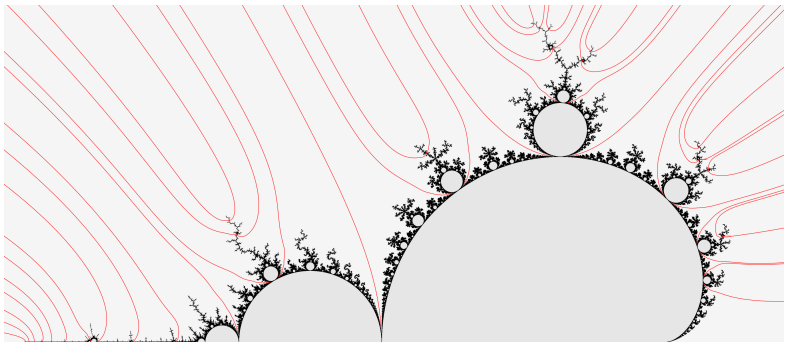


Figure 19: External rays landing at the root of hyperbolic components of periods up to 6.

The third stage is the a-posteriori certification of the data, as explained in Section 6.2. It is split in two sub-stages. First, we read the files in batch and count that we have all roots in a strictly increasing lexicographic order. The corresponding apps are `hypCount` and `misCount`. For reliability reasons, this task is performed on a a single CPU and the code is as independent as possible from the rest of the library. Next (and last), the certification involves redoing all the proofs in high-precision disk arithmetic using as many CPUs as possible, which is the role of the apps `hypProve` and `misProve`.

In practice, the main difficulty for the certification of $c \in \mathrm{Hyp}(n)$ is a close transit of $p_k(c)$ near zero for some $k|n$. This phenomenon can hinder the certification of a root of p_n and prevents essentially any certification attempt made with interval arithmetic. On the other hand, disk arithmetic (see Section 5.3) is essentially immune to this problem. Once the proper certification radius (see Section 6.2) was identified, the process went on flawlessly.

For $\mathrm{Mis}(\ell, n)$, we encountered difficulties in the raw search that kept missing pairs of particularly close parameters near the tips of some antennas (see Figure 21). However, adjusting the parameters in Theorem 9 solved the issue.

The overall computation times of all stages are given in Table 2. As the certification has an obvious $O(d)$ complexity, it can be used as a reference time. The ratio of 2.4 (for p_n) or 9.4 (for $q_{\ell,n}$) between the time necessary for the raw search and that necessary for the certification attests that our algorithm behaves in practice as $O(d)$ with a reasonable constant, at least for the p_n and $q_{\ell,n}$ families. The higher raw search time for Misiurewicz parameters is mostly driven by the fact that, asymptotically, half the roots of $q_{\ell,n}$ are high-multiplicity hyperbolic divisors that are discarded in the end. Note that we did not used the simplified polynomials $m_{\ell,n}$, nor $s_{\ell,n} = p_{\ell+n-1} + p_{\ell-1}$. We can therefore claim that, in practice, the presence of high multiplicities roots slows down the search time with our level-line algorithm by a mere factor 1.95.

	Root finding		Certification		Total	Find/certif. ratio
	Parallel jobs	Merge	Counting	Proofs		
Hyperbolic $p_{n,\ n\leq 41}$	26.2 y 229 kh 70.3%	16.1 d 386 h 0.1%	1.7 d 41 h 0.01%	11 y 96.4 kh 29.6%	37.2 y **∼326 kh**	2.4
Misiurewicz $q_{\ell,n,\ \ell+n\leq 35}$	40.9 y 358 kh 89.4%	9.1 d 218 h 0.05%	1.1 d 26 h 0.006%	4.4 y 38.2 kh 10.6%	45.3 y **∼397 kh**	9.4

Table 2: Overal computation time of each main stage. See Footnote 1 p. 4 for a definition of time-core. Note that for Misiurewicz polynomials, we did not use the simplified version based on $m_{\ell,n}$ in order to measure the effects of high-multiplicities (about 50% of the roots for $q_{2,n}$) on our level-line algorithm.

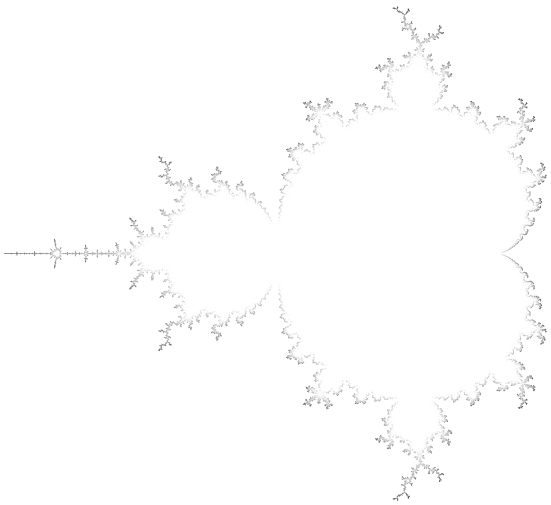


Figure 20: The harmonic measure of $\mathcal{M}^c$ can be visualized by counting how many hyperbolic centers land on each pixel and adjusting the shade accordingly (which is the principle of our `map` files). Here we represent $\mathrm{Hyp}(n)$ for $n \leq 24$. Note the difference with Figure 1 as the inner creeks are visited less frequently than antennaes.

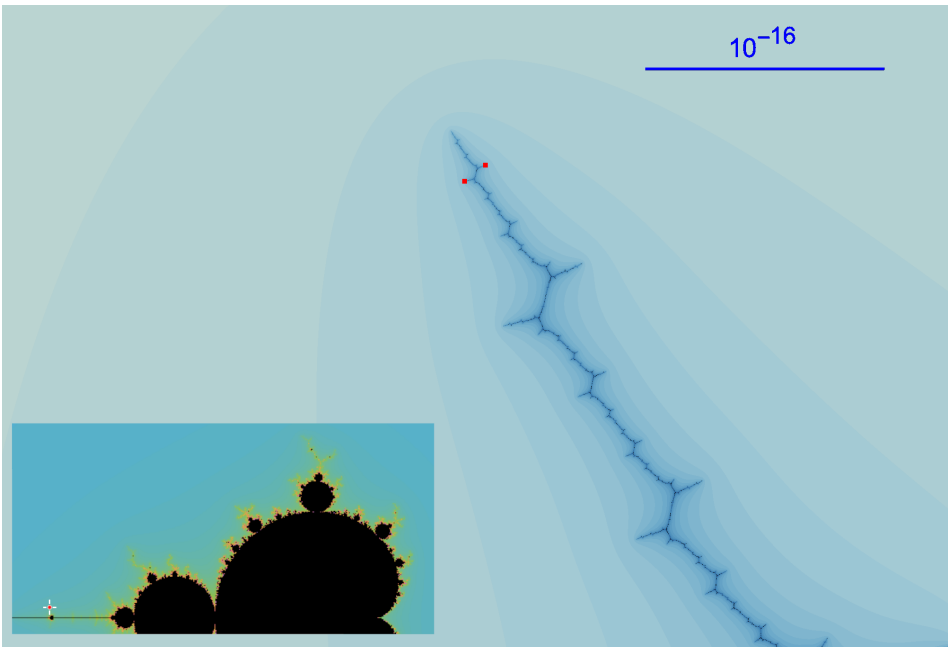


Figure 21: Typical example of a pair of $\mathrm{Mis}(7, 27)$ parameters that are particularly easy to miss during the raw search.

6.4.2 Number of Newton steps and usage of high precision arithmetic

In this subsection, we present some statistics regarding the practical complexity of our algorithm, when applied to the splitting of p_n or $q_{\ell,n}$ when n (resp. $\ell+n$) is large. To normalize the statistics, we present them in terms of *Newton steps per root* or in *Newton steps per new parameter*. Each computable action boils down, eventually, to a Newton steps or equivalent (evaluate a polynomial expression, its derivative and a quotient). We have sorted the actions in categories defined by what is actually being computed and we normalize the total number of Newton steps for each action by the degree of the polynomial (steps per root) or by the cardinal of the set being computed (steps per new parameter). For $\mathrm{Mis}(\ell, n)$ sets, the steps per parameter is roughly double the number of steps per roots because, asymptotically, only half the roots are kept in the set (see Section 2.4).

Period	**Level line**	**Convergent descend**			**Discarded descend**			
n	$\mathbb{L}_{5,8d}(p_n)$	%	`FP80`	`FP128`	Repeat	Overlap	Divisor	Divergent
28	51.6 N/r	26%	8.8 N/p	2.4 N/p	73%	-	0.01%	1%
30	50.7 N/r	25%	8.7 N/p	2.5 N/p	73%	0.07%	0.00%	1%
35	47.9 N/r	25%	8.4 N/p	2.8 N/p	73%	0.23%	0.00%	1%
39	46.9 N/r	25%	8.1 N/p	3.1 N/p	73%	0.38%	0.00%	1%
40	47.2 N/r	25%	7.9 N/p	3.2 N/p	73%	0.39%	0.00%	1%
41	47.7 N/r	25%	7.7 N/p	3.4 N/p	73%	0.34%	0.00%	1%

Table 3: Count of Newton steps per new hyperbolic parameter (N/p), which is equivalent to Newton steps per root (N/r). Intermediary periods have similar statistics.

Type	**Level line**		**Convergent descend**			**Discarded descend**			
$\ell+n$	$\mathbb{L}_{100,8d}(q_{\ell,n})$		%	`FP80`	`FP128`	Repeat	Overlap	Divisor	Divergent
28	110 N/p	56.8 N/r	13%	9 N/p	2.5 N/p	33%	0.02%	48.1%	6%
30	107 N/p	55.4 N/r	13%	8.9 N/p	2.6 N/p	34%	0.01%	48.1%	6%
32	104 N/p	53.9 N/r	12%	8.6 N/p	2.9 N/p	34%	0.00%	48.1%	6%
33	120 N/p	58.8 N/r	13%	10.8 N/p	4.3 N/p	35%	0.01%	49.2%	4%
35	131 N/p	64.4 N/r	13%	10.5 N/p	4.8 N/p	35%	0.02%	49.2%	4%

Table 4: Count of Newton steps per new Misiurewicz parameter (N/p) and Newton steps per root (N/r). The normalization is substantially affected by the deflation due to roots of divisors. In this benchmark, we did not simplify $q_{\ell,n}$ in order to estimate the effect of high-multiplicities on our algorithm.

The bulk of the cost is taken by the Newton steps required to compute the finest level line $\mathbb{L}_{\lambda_0,Md}$ with $\lambda_0 = 5$, $M = 8$, $d = 2^{n-1}$ for p_n and $\lambda_0 = 100$, $M = 8$ and $d = 2^{\ell+n-1}$ for $q_{\ell,n}$. According to Tables 3 and 4, this step appears to have a constant cost of about 50 Newton steps per root, regardless of the degree of the polynomial. This indicate that the computation of the level line is essentially not affected by the presence of roots with high multiplicities. The only exception occurs for $m_{\ell,n}$ and $\ell + n \geq 33$

where a significant jump in the number of steps per roots is observed; it suggests that the level $\lambda_0 = 100$ may be slightly too high for those types.

Next, we perform Newton descends that we initiate on the subset $\mathbb{L}_{\lambda_0,4d}$ (*i.e.* we have 4 starting points per root). Each Newton descend can either end up as convergent towards a new parameter (which is kept in the current `raw*.nset` file) or it can be discarded for one the following reasons:

- the descend converges towards a root that was already found in the current job,
- the descend converges towards a root that overlaps a neighboring job (early detection of duplicates),
- the descend converges towards a root of a divisor,
- the descend is considered divergent and is interrupted, either because of a long-distance jump (indicating the likely proximity of a critical point) or because it did not converge in the allotted number of steps (timeout).

Again, we observe consistently that convergent descends take about 10-12 steps, most of which can be performed entirely using hardware arithmetic. If one assumes that the level line $p_n(z) = 10^{-5}$ is totally disconnected (*i.e.* it has one connected component in each contracting Newton disk), the general considerations of Section 4 thus ensure a practical upper bound of $\log 5 \times 10^5 \sim 13$ Newton steps (up to some universal constant) during the descend from the level line $p_n(z) = 5$. This prediction is conform with our observations.

We have consistently observed that about 5 to 7 % of the roots of p_n are real ones. For $m_{\ell,n}$, the proportion is asymptotically the same for large $\ell + n$, even though the proportion starts at a higher level (9 to 15% for $10 \leq \ell + n \leq 20$).

Tables 3 and 4 indicate that the number of Newton steps for each stage of our algorithm is relatively stable and scales linearly with the degree of the polynomial. However, we have observed an extreme variability in job durations, even though each job is calibrated to deal with the same number of points on the level line. It turns out that the duration is mostly determined by the amount of computations of the level line that need to be performed in high-precision arithmetic, with the `MPFR` library. For our implementation, a practical rule of thumb is that the duration, in hour, of a large job is about $14 \times R + 10$ where R is the ratio of computations that require high-precision. This ratio is illustrated on Figure 22. For period 33, most level lines can be completed using hardware arithmetic. This proportion drops to 12% on average for splitting p_{41}, with some jobs (typically near the left-most tip, see Section 6.3.2) requiring almost exclusively high-precision computations. Note that, on the other hand, the proportion of `MPFR` computations in the Newton descend is constant.

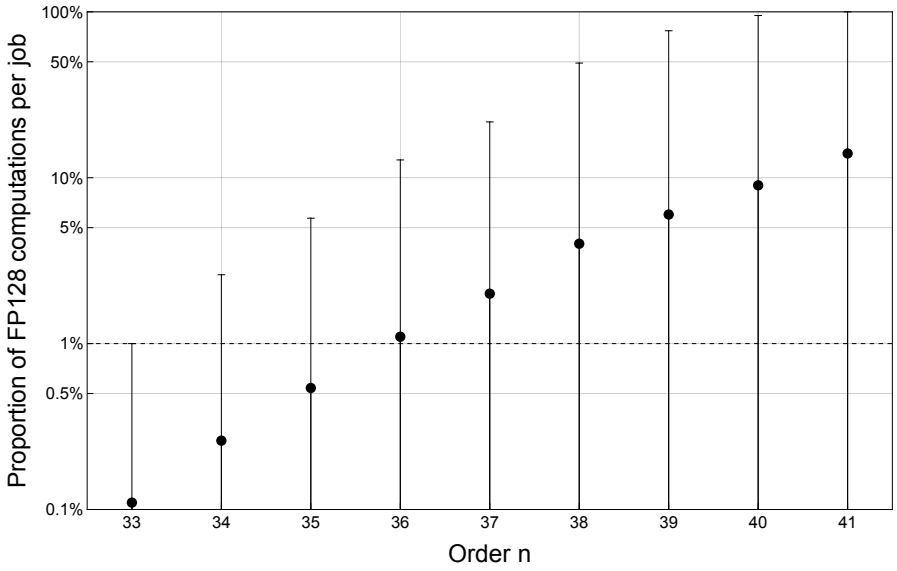


Figure 22: Proportion of the computations of the level fine used for splitting p_n that were performed using high-precision arithmetic operations [MPFR]. The black dot represents the average value. The error bars represent the variability between all raw search jobs.

Overall, the benchmarks presented in this section confirm that, even in the presence of roots with high multiplicities, our algorithm based on level lines is really robust and performs well. Its practical complexity remains $O(d)$, even when d is pushed to the tera-scale. Those observations concern the family of polynomials related to the Mandelbrot set and we do not claim nor suggest that those exceptional performances will remain in full generality.

6.4.3 Memory and hardware requirements

The raw search of roots was split into many independent jobs. Jobs where dimensioned so that they would, on average, correspond to 1.1GB of data with a few exceptions near $z = -2$ where the job size can raise up to a little more than 2GB. Two reasons prevailed in favor of that choice.

First, the memory requirement for each job is fairly limited: each job compares the new roots with the previous one (if pressent) so both are loaded in memory and eat up 4GB, plus the internal data structure can take up another 2GB. Overall, we were able to run all our computations while requiring only 7GB of RAM per CPU. This limit happens to roughly coincide with the average of RAM available on our cluster, when the total RAM of each computing node is equally divided among all the cores of the node. The practical consequence was that the scheduler was usually able to accommodate our requests for a high number of parallel cores, which would not have been the case if a few CPU would have eaten up the whole memory of a node.

Second, writing the final data back into a few files of 1TB did not strike us a good idea, because few people have access to resources that allow them to handle such a monster file reliably, while transferts of 2G files are common.

The second stage (counting that we have all roots and write the final files without duplicates) is performed on a single CPU and requires as much memory as possible. For hyperbolic periods 37 and up, we used 175GB of RAM which allowed most sort operations to be performed without reloading the same file twice.

Our implementation can accommodate tight memory limitations, though the time required to sort and merge the entire database may improve significantly if multiple reloads are necessary. For example, for Misiurewicz types of order $\ell + n \geq 32$, we had to reduce the memory per core to accommodate the growing number of subtypes[12]

Note that the loading time (waiting for `fopen` system calls) can be substantial. In our case, depending on the ambient load on the file system, the loading times ranged from 18% to 98% of the total computation time for the second stage. In some cases, we where able to preload the caches with the upcoming files by running a side process that would monitor the logs and `cat file > /dev/null` while the previous files where being processed.

The third stage (certification) requires very little memory (no more than twice size of a final `nset` file) so 5GB per CPU dedicated to a certification task is sufficient.

6.4.4 The jobs market

The raw search stage is embarrassingly parallel. However, the duration of the jobs presents large variability even for a given hyperbolic period or a Misiurewicz sub-type: it can range from half the average compute time up to 8 times the average duration of the jobs that handle the same polynomial. This heterogeneity calls for a vertical parallelism, where each compute node calls for a new task as soon as it becomes available, asynchronously from the other nodes that remain busy.

The human task of submitting new jobs is also non trivial: to generate our database, even without a single failure, 49 427 raw jobs are necessary (about 2/3 of them for $\mathrm{Hyp}(n)$), followed by 1202 jobs for the later phases (most of them for $\mathrm{Mis}(\ell, n)$). Identifying the raw jobs that have crashed due to node failure, network saturation, disk lag (which happened to be a common plague on our busy HPC center) or job timeout can easily turn into a nightmare.

To optimise our use of HPC resources among heterogeneous jobs and automate the task of submitting raw jobs, we developed our own task scheduler, which is a small `Java` app, called *job market*. It records the progress of all computing tasks, distributes new tasks asynchronously to each available computing node, recycles failed jobs and produces useful statistics.

[12] For $\mathrm{Mis}(\ell, n)$ with $\ell + n = 33$, we requested only 26GB for each of the 31 cores dedicated to the count, instead of 73GB for the single core used for Hyp(33). Consequently, each file had to be reloaded about 2.5 times.

6.4.5 Use a GPU or not?

In order to keep our implementation as simple (and human readable) as possible, we decided against a mixed code GPU/CPU. However, we may revisit this position in the future. Here are a few thoughts about this choice.

For massively parallel computations, GPUs offers impressive performances. We briefly explored this path; however, our algorithm reaches the limits of hardware arithmetic very quickly (see §6.3.2). For now, GPUs are essentially limited to FP64 arithmetic, which gives a first objection against their usage for splitting polynomials of high degree, pending a library that can emulate high-precision arithmetic.

The second objection is that the computational gain is not completely obvious because it is not easy to predict how long the dynamics of the Newton map will take before the algorithm can classify the starting point. It is silly to keep running Newton steps just because other threads are still doing it; and it is equally counterproductive to cut an iteration short just because the others are done. Considering that the main gain of GPUs lies in running identical parallel threads, a naive implementation may therefore tend to impose the worst case as the statistical norm, thus defeating the purpose of running on a GPU.

Appendices

A Proof of the factorization theorem

Let us give here a direct proof of the factorization theorem (*i.e.* Theorem 2).

Proof. As mentioned above, thanks to [HT15], only the multiplicity of the hyperbolic factors has to be established. For $\ell \in \{0, 1\}$, the formula (13) boils down to (8) so we will suppose $\ell \geq 2$ from now on.

The case of $\mathrm{Hyp}(1) = \{0\}$ is essentially based on the well known fact (proven by direct induction) that the trailing coefficients of $p_n(z)$ stabilize as n grows; more precisely:

$$\forall k \in \mathbb{N}, \qquad p_n(z) \equiv p_{n+k}(z) \mod z^{n+1}, \tag{50}$$

thus $q_{\ell,n}(z) \equiv 0 \mod z^{\ell+1}$. On the other hand, one can check that

$$q_{\ell,n}(z) \equiv 2^{\ell-1} z^{\ell+1} \mod z^{\ell+2} \tag{51}$$

so zero is exactly of multiplicity $\ell + 1 = \eta_\ell(1)$.

Next, let us check that the multiplicities in (13) are consistent with $\deg q_{\ell,n}$, *i.e.* :

$$\sum_{k|n} \left(\eta_\ell(k) \, |\mathrm{Hyp}(k)| + \sum_{j=2}^{\ell} |\mathrm{Mis}(j,k)| \right) = 2^{\ell+n-1}. \tag{52}$$

Indeed, using (7) and (12) and a geometric sum, the left-hand side equals

$$\sum_{k|n} |\mathrm{Hyp}(k)| \left(\eta_\ell(k) + \sum_{j=2}^{\ell} \Phi(j,k) \right)$$
$$= \sum_{k|n} \left(\sum_{m|k} \mu(k/m) 2^{m-1} \right) \left(2^\ell + \left\lfloor \frac{\ell-1}{k} \right\rfloor - \sum_{j=2}^{\ell} \delta_{k|j-1} \right)$$

where $\delta_{k|j-1} = 1$ if k divides $j-1$ and 0 otherwise. For any integers $\lambda, k \in \mathbb{N}^*$, let us observe that

$$\left\lfloor \frac{\lambda}{k} \right\rfloor = |\{ j \in [\![1, \lambda]\!] \, ; \, k|j \}| = \sum_{j=1}^{\lambda} \delta_{k|j}. \tag{53}$$

The claim (52) thus follows from Möbius inversion formula [Möb32]-[Rot63]:

$$\sum_{k|n} \left(\sum_{m|k} \mu(k/m) 2^m \right) = 2^n. \tag{54}$$

Let us now consider the case of $\mathrm{Hyp}(k)$ for $k|n$ and $1 < k \leq n$. Note that if n is prime, there is only one hyperbolic factor left, namely $k = n$ and the factorization (13) follows from an argument of divisibility and the identity of degrees (52). We can however treat the general case in a unified way, regardless of wether n is composite or not and prove (13) by recurrence on ℓ. One has

$$\begin{aligned} q_{\ell,n}(z) &= p_{\ell+n}(z) - p_\ell(z) \\ &= p_{\ell+n-1}^2(z) - p_{\ell-1}^2(z) \\ &= q_{\ell-1,n}(z)(p_{\ell+n-1}(z) + p_{\ell-1}(z)) \\ &= q_{\ell-1,n}(z)(q_{\ell-1,n}(z) + 2p_{\ell-1}(z)). \end{aligned} \tag{55}$$

The recurrence assumption reads

$$q_{\ell-1,n} = \prod_{k|n} \left(h_k^{\eta_{\ell-1}(k)} \prod_{j=2}^{\ell-1} m_{j,k} \right) \tag{56}$$

and, from the hyperbolic case, we know that

$$p_{\ell-1} = \prod_{j|\ell-1} h_j.$$

If k divides n but not $\ell-1$, then h_k divides $q_{\ell-1,n}(z)$ but not $p_{\ell-1}$ so h_k does not divide $q_{\ell-1,n} + 2p_{\ell-1}$. Alternatively, if k divides both n and $\ell-1$ then h_k divides $q_{\ell-1,n} + 2p_{\ell-1}$. However h_k^2 does not because it is a factor of $q_{\ell-1,n}$ (see the Lemma 16 below) but not

of $p_{\ell-1}$. Finally, the polynomials $m_{\ell,k}$ for $k|n$ are known factors of $q_{\ell,n}$ that do not divide $q_{\ell-1,n}$; as $q_{\ell,n}/q_{\ell-1,n}$ has simple roots, they are simple factors of $q_{\ell,n}$. There are no other Misiurewicz-type factors. We have thus established that:

$$q_{\ell,n} = \left(\prod_{k|\gcd(n,\ell-1)} h_k\right)\left(\prod_{k|n} m_{\ell,k}\right) q_{\ell-1,n}. \tag{57}$$

Let us finally observe that, for $\ell, k \geq 2$:

$$\eta_{\ell-1}(k) = \begin{cases} \eta_\ell(k) - 1 & \text{if } k|\ell-1, \\ \eta_\ell(k) & \text{else.} \end{cases} \tag{58}$$

Indeed, if $k|\ell-1$ then (58) follows directly from (14); conversely, if k does not divide $\ell-1$ then $\frac{\ell-1}{k} \in \mathbb{Z} + [\frac{1}{k}, 1)$ and

$$\eta_{\ell-1}(k) = \left\lfloor \frac{\ell-1}{k} - \frac{1}{k} \right\rfloor + 2 = \left\lfloor \frac{\ell-1}{k} \right\rfloor + 2 = \eta_\ell(k).$$

Combining (56), (57) and (58) ensures that (13) holds for the next pre-period. □

Remark 15. *The previous proof establishes the following identity (see also (9)):*

$$s_{\ell,n} = \frac{q_{\ell,n}}{q_{\ell-1,n}} = p_{\ell+n-1} + p_{\ell-1} = \left(\prod_{k|\gcd(n,\ell-1)} h_k\right)\left(\prod_{k|n} m_{\ell,k}\right). \tag{59}$$

To illustrate our method, the simplest case of a composite period is:

$$q_{2,4}(z) = p_6(z) - p_2(z) = p_5^2(z) - p_1^2(z) = q_{1,4}(z)(p_5(z) + p_1(z)) = p_4^2(z)(p_4^2(z) + 2z).$$

As $p_4 = h_4 h_2 h_1$, the polynomial $q_{2,4}$ is divisible by h_4^2 but not by h_4^3. The exponent of Hyp(4) is therefore exactly 2 and we are left with a single hyperbolic factor, namely Hyp(2). Using the degree identity (52) thus ensures that

$$q_{2,4}(z) = h_1^3(z) h_2^2(z) h_4^2(z) m_{2,1}(z) m_{2,2}(z) m_{2,4}(z).$$

The only check-up left is the following statement.

Lemma 16. *If $k \geq 1$ divides both $\ell \geq 1$ and $n \geq 1$, then h_k^2 is a factor of $q_{\ell,n}$.*

Proof. It is sufficient to establish the lemma for $q_{k,jk}$ with $j \in \mathbb{N}$ because

$$\forall r, q \in \mathbb{N}^*, \quad q_{kr,kq} = p_{k(q+r)} - p_k - p_{kr} + p_k = q_{k,k(q+r-1)} - q_{k,k(r-1)}.$$

As $q_{k,0} = 0$, let us assume that $j \geq 1$; using (55) repeatedly one has.

$$\begin{aligned} q_{k,jk} &= q_{k-1,jk} \times (p_{(j+1)k-1} + p_{k-1}) \\ &= q_{k-2,jk} \times (p_{(j+1)k-2} + p_{k-2})(p_{(j+1)k-1} + p_{k-1}) \\ &= \dots \\ &= q_{1,jk} \times (p_{(j+1)k-2} + p_{k-2})(p_{(j+1)k-1} + p_{k-1}) \dots (p_{jk+1} + p_1) \end{aligned}$$

Therefore, $p_{jk}^2 = q_{1,jk}$ is a factor of $q_{k,jk}$. As $k|jk$, then h_k is itself a factor of p_{jk} and so h_k^2 divides p_{jk}^2 and consequently $q_{k,jk}$ too. □

B Useful mathematical results

We expect our readers to have (like us authors) varied mathematical backgrounds. For the convenience of all, we recall here briefly a few mathematical results that are quite standard to the specialists, but might not be well known to all.

B.1 Localization of the roots of polynomials

To renormalize the roots of $P \in \mathbb{C}[z]$ within $\mathrm{D}(0,1)$ one may consider $P(z/\rho)$ where $\rho > r$ and r is given by the following statement.

Theorem 10 ([BE95, Theorem 1.2.4])**.** *All the zeros of* $P(z) = \sum_{k=0}^{n} a_k z^k$ *are located in the disk* $\overline{\mathrm{D}(0,r)}$ *where*

$$r = \inf_{\frac{1}{p}+\frac{1}{q}=1} \left\{ 1 + \left(\sum_{j=0}^{n-1} \frac{|a_j|^p}{|a_n|^p} \right)^{q/p} \right\}^{1/q}$$

and provided $a_n \neq 0$ *and* $p, q > 1$.

Remark 17. *For various classical results on this mater, see [Hen74, §6.4].*

Small perturbations of analytic functions do not affect much the location of their roots, as stated by Rouché's theorem, stated here in its strong symmetric form.

Theorem 11 (Rouché-Estermann)**.** *Given two holomorphic functions* f, g *on a domain* G *and a bounded region* $K \subset G$ *with continuous boundary* ∂K*. If the following strict inequality holds:*

$$\forall z \in \partial K, \qquad |f(z) - g(z)| < |f(z)| + |g(z)|$$

then f *and* g *have the same number of roots (counted with multiplicity) in* K*.*

A common case of application is when $|h(z)| < |f(z)|$ on ∂K. In that case, the theorem can be applied with $g = f + h$ and ensures that the perturbation h does not change the number of roots of f.

B.2 Injectivity of analytic functions

The following result quantifies the local injectivity of analytic functions.

Theorem 12 (Koebe's lemma)**.** *If* $f : \mathrm{D}(0,1) \to \mathbb{C}$ *is an injective analytic function, then*

$$f(\mathrm{D}(0,1)) \supset \mathrm{D}(f(0), |f'(0)|/4).$$

The universality of the constant $1/4$ constitutes a radical breach between real and complex analysis. The optimality of this constant can be checked on the function

$$f(z) = \frac{z}{(1-z)^2}$$

because $f'(0) = 1$ but the value $f(-1) = -1/4 \notin f(\mathrm{D}(0,1))$.

One simple criterion that ensures injectivity is $0 \notin f'(\mathrm{D}(0,1))$ *i.e.* the absence of critical point within $\mathrm{D}(0,1)$.

C Index of notations

We provide here a short index of our notations. By default, we use the American standard names, notations and spellings.

About integer, real and complex numbers

$k|n$: the integer k is a divisor of the integer n.
$\mathrm{Div}(n)$: set of all divisors of $n \in \mathbb{N}^*$
$\mathrm{Div}(n)^* = \mathrm{Div}(n)\backslash\{n\}$: strict divisors of n

Floor function: $\lfloor x \rfloor = k$ if $k \in \mathbb{Z}$ and $x \in [k, k+1)$
$\mathrm{I}(x,r) = (x-r, x+r)$: open interval along the real line.

$z = a + ib \in \mathbb{C}$: complex numbers (with $a, b \in \mathbb{R}$)

$$|z| = \sqrt{x^2+y^2} \qquad \text{and} \qquad |z|_1 = |x| + |y|.$$

$\mathrm{D}(z,r) = \{z' \in \mathbb{C}\,;\, |z'-z| < r\}$: complex open disk.
$\overline{\mathbb{C}} = \mathbb{C} \cup \{\infty\}$: the Riemann sphere.
$\mathbb{U}_N = \left\{e^{2ik\pi/N}\,;\, 0 \le k < N\right\}$: the roots of unity of order $N \in \mathbb{N}^*$.
$\mathring{\Omega}$, $\overline{\Omega}$: interior and closure of a subset $\Omega \in \overline{\mathbb{C}}$.

About the Mandelbrot set

$f_c(z) = z^2 + c$: the fundamental map (3) for the dynamics associated to $\mathcal{M}$.
p_n and $q_{\ell,n}$: hyperbolic (1) and Misiurewicz-Thurston (2) polynomials.
$\mathcal{M} = \{c \in \mathbb{C}\,;\, \forall n \in \mathbb{N},\ |p_n(c)| \le 2\}$: Mandelbrot set (Figure 1).
$\mathrm{Hyp}(n)$: set of hyperbolic centers of order n, defined by (4).
$\mathrm{Mis}(\ell, n)$: set of pre-periodic (Misiurewicz-Thurston) parameters of type (ℓ, n), defined by (10).
h_n, $m_{\ell,n}$ and $s_{\ell,n}$: reduced polynomials (5), (11) and (9).

About polynomials and splitting algorithms

$\mathrm{Crit}(P)$: Set of critical points of P, *i.e.* $P'(z) = 0$.
N_P : Newton map associated with a polynomial P (see Section 3.4).
$\zeta(t)$: Newton's flow defined by the ODE (24).
$\lambda(t)$: Level line defined by the ODE (27).
$\mathbb{L}_{\lambda_0,N}(P)$: Discrete level line of P defined by (31) and central to our splitting algorithm.
ε_R, ε_N, ε_S : various radii related to the certification process (see Theorem 9).
$\mathbb{M}_r(d)$: Universal mesh of [HSS01] for splitting a polynomial of degree d (see Section 3.5).

About finite precision arithmetic

$\mathcal{Z}_N$: fundamental set (40) of floating points numbers with finite precision
$\hat{\mathcal{R}}_N$: (theoretical) set of all (39) floating points numbers with a given precision
$\mathcal{R}_N$: (realistic) subset (41) of $\hat{\mathcal{R}}_N$ with limited exponents
$e(x)$ and $\widehat{\mathrm{ulp}}(x) = 2^{e(x)-N}$: exponent of $x \in \hat{\mathcal{R}}_N \backslash \{0\}$ and associated *unit on last position* (42).
$\hat{\mathrm{R}}_N : \mathbb{R} \to \hat{\mathcal{R}}_N$: rounding operator from the real line onto finite precision numbers.

D Listing of tasks implemented in [Mandel]

In our implementation [Mandel], the tasks listed in this section are called in the command line with `Mandelbrot -task [arguments]`. Use `-task -help` for more detailed informations. The core functions of the interface are listed here. As this software is currently under development, new features will be released regularly.

Tools for the inital setup[13]

`-levelSets`	prepares the level sets for roots search
`-misSets`	prepares the level sets for Misiurewicz points search
`-misSimpleSets`	prepares the simple level sets for Misiurewicz points search

Tools for splitting p_n and $q_{\ell,n}$ using `FP80` hardware arithmetic

`-hypQuick`	computes hyperbolic centers quickly, with low precision
`-misQuick`	computes pre-periodic points quickly, with low precision
`-misSimpleQuick`	computes pre-periodic points quickly, with low precision, using simplified polynomials $s_{\ell,n} = p_{\ell+n-1} + p_{\ell-1}$

Tools for splitting p_n using certifiable `FP128` arithmetic

`-hypRaw`	computes the hyperbolic centers, saves results in ∼1GB binary files
`-hypRawCount`	counts the results of `hypRaw`
`-hypCount`	counts and checks the unicity of hyperbolic centers
`-hypProve`	re-proves the hyperbolic centers and the convergence of the Newton map

Tools for splitting $q_{\ell,n}$ using certifiable `FP128` arithmetic

`-misRaw`	computes the pre-periodic points, saves results in ∼1GB binary files
`-misSimpleRaw`	computes pre-periodic points using simplified polynomials $s_{\ell,n}$
`-misRawCount`	counts the results of `misRaw`
`-misCount`	counts and checks the unicity of Misiurewicz parameters
`-misProve`	re-proves the Misiurewicz points and the convergence of the Newton map

Statistical and graphical tools for analyzing the database

`-hypMinDist`	computes minimum distance between hyperbolic centers
`-misMinDist`	computes minimum distance between pre-periodic points
`-hypTree`	computes the maps of the hyperbolic centers with different resolutions
`-misTree`	computes the maps of the Misiurewicz points with different resolutions
`-bitmap`	renders the bitmaps described in the input file

[13] If possible, those three tasks will be merged into one common interface in future versions.

Tools for handling `csv` **files and our custom formats** `nset` **and** `mpv`	
`-header`	explains the header of a `nset` file and check for basic file integrity
`-nset2csv`	exports data from binary `nset` files to `csv` files
`-csvCompare`	compare `csv` files with numerical values
`-csv2mpv`	packs numbers from `csv` files to binary `mpv` files
`-mpv2mpv`	exports and converts data from binary `mpv` files to `mpv` files
`-mpvCompare`	compare the values stored in two binary `mpv` files

References *

[Abe73] O. Aberth. Iteration methods for finding all zeros of a polynomial simultaneously,. *Math. Comput.*, 27(122):339–344, 1973.

[ACE09] J.T. Albrecht, C.P. Chan, and A. Edelman. Sturm sequences and random eigenvalue distributions. *Foundations of Computational Mathematics*, 9:461–483, 2009.

[AMV22] R. Anton, N. Mihalache, and F. Vigneron. Fast evaluation of complex polynomials. arXiv:2211.06320, 2022.

[AMV23] R. Anton, N. Mihalache, and F. Vigneron. A short ODE proof of the fundamental theorem of algebra. *The Mathematical Intelligencer*, 2023.

[BAS16] T. Bilarev, M. Aspenberg, and D. Schleicher. On the speed of convergence of Newton's method for complex polynomials. *Math. Comput.*, 85(298):693–705, 2016.

[BDG04] D. A. Bini, F. Daddi, and L. Gemignani. On the shifted QR iteration applied to companion matrices. *Electronic Transactions on Num. Analysis*, 18:137–152, 2004.

[BE95] P. Borwein and T. Erdélyi. *Polynomials and Polynomial Inequalities*. Springer, 1995.

[Ber93] W. Bergweiler. Iteration of meromorphic functions. *Bull. AMS*, 29(2):151–188, 1993.

[BH03] X. Buff and C. Henriksen. On König's root finding algorithms. *Nonlinearity*, 16(3):989, 2003.

[Bin96] D. A. Bini. Numerical computation of polynomial zeros by means of aberth's method. *Numerical Algorithms*, 13:179–200, 1996.

[BM92] C. Bernardi and Y. Maday. *Approximations spectrales de problème aux limite elliptiques*. Springer, 1992.

[BMF12] I. Bogaert, B. Michiels, and J. Fostier. O(1) computation of legendre polynomials and gauss–legendre nodes and weights for parallel computing. *SIAM J. Sci. Comput.*, 34:C83–C101, 2012.

[Bre17] J. Bremer. On the numerical calculation of the roots of special functions satisfying second order ordinary differential equations. *SIAM Journal on Scientific Computing*, 39(1):A55–A82, 2017.

[BS05] D. Beliaev and S. Smirnov. Harmonic measure on fractal sets. In European Mathematical Society, editor, *4ECM Stockholm 2004*, pages 41–59, 2005.

[Buf18] X. Buff. On postcritically finite unicritical polynomials. *New York Journal of Mathematics*, 24:1111–1122, 2018.

* Numerical libraries are typeset `[lib]` and are listed at the end of the bibliography.

[Cam19] T.R. Cameron. An effective implementation of a modified laguerre method for the roots of a polynomial. *Numerical Algorithms*, 82:1065–1084, 2019.

[CG93] L. Carleson and T.W. Gamelin. *Complex dynamics*. Springer, 1993.

[CGXZ07] S. Chandrasekaran, M. Gu, J. Xia, and J. Zhu. A fast QR algorithm for companion matrices. *Recent Advances in Matrix and Operator Theory*, 179:111–143, 2007.

[Che10] A. Cheritat. L'ensemble de mandelbrot. *Images des Mathématiques*, https://images.math.cnrs.fr/L-ensemble-de-Mandelbrot.html, 2010.

[DH82] A. Douady and J.H. Hubbard. Itération des polynômes quadratiques complexes. *C.R. Acad. Sci. Paris, Sér. I Math.*, 294(3):123–126, 1982.

[DH85] A. Douady and J.H. Hubbard. *Etude dynamique des polynômes complexes*. Prépublications mathématiques d'Orsay, 1984-1985.

[DT93] E.R. Davidson and W.J. Thompson. Monster matrices: their eigenvalues and eigenvectors. *Computers in Physics*, 7(5):519–522, 1993.

[Fat19] P. Fatou. Sur les équations fonctionnelles. *Bull. SMF*, 47:161–271, 1919.

[FG15] C. Favre and T. Gauthier. Distribution of postrictically finite polynomials. *Israel Journal of Mathematics*, 2015.

[Fra89] P. Fraigniaud. Analytic and asynchronous root finding methods on a distributed memory multi- computer. *Research Report LIP-IMAG*, 1989.

[FS89] M.L. Fredmann and M.E. Saks. The cell probe complexity of dynamic data structures. In *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing*, STOC' 89, pages 345–354. Association for Computing Machinery, 1989.

[GMSB16] A. Gholami, D. Malhotra, H. Sundar, and G. Biros. Fft, fmm, or multigrid? a comparative study of state-of-the-art poisson solvers for uniform and nonuniform grids in the unit cube. *SIAM J. Sci. Comput.*, 38(3), 2016.

[GSCG17] K. Ghidouche, A. Sider, R. Couturier, and C. Guyeux. Efficient high degree polynomial root finding using gpu. *Journal of Computational Science*, 18:46–56, 2017.

[GSKC16] K. Ghidouche, A. Sider, L.Z. Khodja, and R. Couturier. Two parallel implementations of Ehrlich-Aberth algorithm for root-finding of polynomials on multiple GPUs with OpenMP and MPI. In *Intl Conference on Computational Science and Engineering*, 2016.

[Guc] Karol Guciek. https://guciek.github.io/web_mandelbrot.html.

[Gug86] H. Guggenheimer. Initial approximations in durand-kerner's root finding method. *BIT Numerical Mathematics volume*, 26:537–539, 1986.

[GV17] T. Gauthier and G. Vigny. Distribution of postrictically finite polynomials II: speed of convergence. *American Institute of Mathematical Science*, 11:57–98, 2017.

[GV19] T. Gauthier and G. Vigny. Distribution of postrictically finite polynomials III: combinatorial continuity. *Fundamenta Math*, 244(1):17–48, 2019.

[Hen74] P. Henrici. *Applied and computational complex analysis, Volume 1: Power series, integration, conformal mapping, location of zeros*. Wiley, 1974.

[HSS01] J.H. Hubbard, D. Schleicher, and S. Sutherland. How to find all roots of complex polynomials by Newton's method. *Invent. math.*, 146:1–33, 2001.

[HT13] N. Hale and A. Townsend. Fast and accurate computation of gauss-legendre and gauss-jacobi quadrature nodes and weights. *SIAM J. Sci. Comput.*, 35:A652–A674, 2013.

[HT15] Benjamin Hutz and Adam Towsley. Misiurewicz points for polynomial maps and transversality. *New York Journal of Mathematics*, 21:297–319, 2015.

[IEE] IEEE 754. https://en.wikipedia.org/wiki/IEEE_754.

[IYM11] T. Imamura, S. Yamada, and M. Machida. Development of a high-performance eigensolver on a peta-scale next-generation supercomputer system. *Progress in Nuclear Science and Technology*, 2:643–650, 2011.

[Jun85] I. Jungreis. The uniformisation of the complement of the Mandelbrot set. *Duke Math. J.*, 52(4):935–938, 1985.

[KI04] N. Kyurkchiev and A. Iliev. Failure of convergence of the newton-weierstrass iterative method for simultaneous rootfinding of generalized polynomials. *Computer and Mathematics with Applications*, 47:441–446, 2004.

[KS94] M.-H Kim and S. Sutherland. Polynomial root-finding algorithms and branched covers. *SIAM Journal on Computing*, 23(2):415–436, 1994.

[KS16] A. Kobel and M. Sagraloff. Fast approximate polynomial multipoint evaluation and applications. arXiv:1304.8069, 2016.

[Lar13] K.G. Larsen. *Models and Techniques for Proving Data Structure Lower Bounds Models and Techniques for Proving Data Structure Lower Bounds Models and Techniques for Proving Data Structure Lower Bounds*. PhD thesis, Aarhus University, Denmark, 2013.

[Lev90] G.M. Levin. On the theory of iterations of polynomial families in the complex plane. *J. Soviet Math.*, 52(6):3512–3522, 1990.

[Mak85] N.G. Makarov. On the distortion of boundary sets under conformal mappings. *Proceedings of the London Mathematical Society*, 51(2):369–384, 1985.

[Mil90] J. Milnor. *Dynamics in one complex variable*. Number 160 in Annals of Mathematics Studies. Princeton Univ. Press, 1990.

[Möb32] A.F. Möbius. Über eine besondere art von umkehrung der reihen. *Journal für die reine und angewandte Mathematik*, 9:105–123, 1832.

[Mor13] G. Moroz. Fast polynomial evaluation and composition. Technical Report 453, Inria Nancy - Grand Est, LORIA - ALGO - Department of Algorithms, Computation, Image and Geometry, 2013.

[MPRW22] D. Martí-Pete, L. Rempe, and J. Waterman. Bounded Fatou and Julia components of meromorphic functions. arXiv:2204.11781, 2022.

[OEIS] Online Encyclopedia of Integer Sequence. Sequence A000740. https://oeis.org/A000740, 1991.

[Pan02] V.Y. Pan. Univariate polynomials: Nearly optimal algorithms for numerical factorization and root-finding. *J. Symbolic Computation*, 33:701–733, 2002.

[PP98] M.S. Petković and L.D. Petković. *Complex interval arithmetic and its applications*, volume 105. Wiley-VCH, 1998.

[RAY19] K. Ravikumar, D. Appelhans, and P.K. Yeung. Gpu acceleration of extreme scale pseudo-spectral simulations of turbulence using asynchronism. Technical report, The International Conference for High Performance Computing, Networking, Storage and Analysis, DOI: 10.1145/3295500.3356209, 2019.

[Rok01] J.G. Rokne. *Interval Arithmetic and Interval Analysis: An Introduction.* in book Granular Computing: An Emerging Paradigm, 2001.

[Rot63] G.-C. Rota. On the foundations of combinatorial theory, I: Theory of Möbius functions. *Z. Wahrscheinlichkeitstheorie u. verw. Gebiete*, 2:340–368, 1963.

[RR05] N. Revol and F. Rouillier. Motivations for an arbitrary precision interval arithmetic and the MPFI library. *Reliable Computing*, 11:275–290, 2005.

[RSS17] M. Randig, D. Schleicher, and R. Stoll. Newton's method in practice II: The iterated refinement Newton method and near-optimal complexity for finding all roots of some polynomials of very large degrees. arXiv:1703.05847, 2017.

[RSS20] B. Reinke, D. Schleicher, and M. Stoll. The weierstrass root finder is not generally convergent. ArXiv:2004.04777, 2020.

[Sch23] D. Schleicher. On the efficient global dynamics of newton's method for complex polynomials. *Nonlinearity*, 36:1349–1377, 2023.

[SCR+20] S. Shemyakov, R. Chernov, D. Rumiantsau, D. Schleicher, S. Schmitt, and A. Shemyakov. Finding polynomial roots by dynamical systems – a case study. *Discrete and Continuous Dyn. Systems*, 40(12):6945–6965, 2020.

[Sib84] N. Sibony. Exposés à Orsay non publiés & Cours UCLA. 1981-1984.

[SK19] Angelika Schwarz and Lars Karlsson. Scalable eigenvector computation for the non-symmetric eigenvalue problem. *Parallel Computing*, 85:131–140, 2019.

[SS17] D. Schleicher and R. Stoll. Newton's method in practice: finding all roots of polynomials of degree one million efficiently. *Theor. Comput. Sci.*, 681:146–166, 2017.

[Sut89] S. Sutherland. *Finding root of complex polynomials with Newton's method.* PhD thesis, Boston University, 1989.

[SZI+17] T. Sakurai, S.-L. Zhang, T. Imamura, Y. Yamamoto, Y. Kuramashi, and T. Hoshi, editors. *Eigenvalue Problems: Algorithms, Software and Applications in Petascale Computing*, volume 117 of *Lecture Notes in Comp. Science and Engineering.* Springer, 2017.

[TAB+19] P.-H. Tournier, I. Aliferis, M. Bonazzoli, M. de Buhan, M. Darbas, V. Dolean, F. Hecht, P. Jolivet, I. El Kanfoud, C. Migliaccio, F. Nataf, Ch. Pichot, and S. Semenov. Microwave tomographic imaging of cerebrovascular accidents by using high-performance computing. *Parallel Computing*, pages 88–97, 2019.

[TTO16] A. Townsend, T. Trogdon, and S. Olver. Fast computation of gauss quadrature nodes and weights on the whole real line. *IMA J. Numer. Anal.*, 36:337–358, 2016.

[vdH09] Joris van der Hoeven. Ball arithmetic. *Technical report: HAL-00432152.*, 2009.

[Wil84] J. H. Wilkinson. *The perfidious polynomial*, pages 1–28. Studies in Numerical Analysis. G. H. Golub, 1984.

[ARB] F. Johansson. Arb library. https://github.com/fredrik-johansson/arb, 2012.

[FLINT] W. Hart and F. Johansson and S. Pancratz. FLINT: Fast Library for Number Theory. http://flintlib.org, 2013.

[FPE] N. Mihalache and F. Vigneron. FPE library: a Fast Polynomial Evaluator. https://github.com/fvigneron/FastPolyEval, 2022.

[Mandel] N. Mihalache and F. Vigneron. Mandel library: a Numerical Microscope onto the Mandelbrot set. https://github.com/fvigneron/Mandelbrot, 2024.

[Mand.DB] N. Mihalache and F. Vigneron. Complete list of hyperbolic centers of period ≤ 41 and of all Misiurewicz-Thurston points whose pre-period and period sum is ≤ 35.

[MPFR] L. Fousse, G. Hanrot, V. Lefèvre, P. Pélissier and P. Zimmermann. MPFR: a Multiple-Precision binary Floating-point library with correct Rounding. *ACM Trans. Math. Software*, 33(2):13–28. https://www.mpfr.org, 2007.

[1] **Nicolae Mihalache.** Univ Paris-Est Creteil, CNRS UMR 8050, LAMA, F-94010 Creteil, France and Univ Gustave Eiffel, LAMA, F-77447 Marne-la-Vallée, France
nicolae.mihalache@u-pec.fr

[2] **François Vigneron.** Université de Reims Champagne-Ardenne, Laboratoire de Mathématiques de Reims, UMR 9008 CNRS, Moulin de la Housse, BP 1039, F-51687 Reims
francois.vigneron@univ-reims.fr