To Whom It May Concern:

This summer, I attended an REU (Research Experience in Mathematics) at the University of Chicago. At the end of the program, I wrote a paper on cohomology theory (a subfield of pure algebraic topology) and dynamics which are two fields in advanced mathematics. The paper, attached below, develops and presents original research towards a problem studied by Fields Medalist Dr. Steve Smale.

In an influential 1989 paper, Smale defined the *topological complexity* of an algorithm. Simply put, it is one of many ways to measure how slow a computer algorithm is. Smale was particularly interested in the topological complexity of computer algorithms which find the roots of polynomials. In his paper, he proved that *any* algorithm which computes the roots of a polynomial has a minimum topological complexity, and he computed that value (as a function of the degree of the input polynomial).

We provide an upper bound on the topological complexity of an optimal program which computes the roots of a polynomial. The upper bound is explicitly computed for specific types of polynomials (when the degree of the polynomial is 2, 3, or 4) and we make a more general conjecture regarding an upper bound on the topological complexity of the optimal program which computes the roots of any polynomial.

The paper is attached below.

Sincerely,
Parth Sarin

THE CUP LENGTH IN COHOMOLOGY AS A BOUND ON TOPOLOGICAL COMPLEXITY

PARTH SARIN

Abstract. Polynomial solving algorithms are essential to applied mathematics and the sciences. As such, reduction of their complexity has become an incredibly important field of topological research. We present a topological approach to constructing a lower bound for the complexity of a polynomial-solving algorithm, and give a concrete algorithm to do this in the case that $\deg(f) = 2, 3, 4$.

Contents

1. Defining the problem 1
2. Algorithmic processes and nodes 2
3. From computation theory to algebraic topology 3
4. The cup length as a bound on the covering number 5
5. Computing the kernel of π^* 6
6. Conclusion 7
7. Low-complexity algorithms via Newton's Method on the complex plane 7
7.1. Quadratic polynomials 8
7.2. Cubic polynomials 9
7.3. Quartic polynomials 11
7.4. $t^d - S$ 12
8. Low-complexity algorithms via Power Iteration 13
8.1. Equal-magnitude dominant and subdominant eigenvalues 14
Acknowledgments 15
References 15

1. Defining the problem

When computers execute programs to solve polynomials, it is inevitable that such programs include "if" statements. These decisions create branching in the algorithm. We would like to bound the amount of branching in algorithms. Formally, the problem at hand is:

Poly(d): Given a complex monic polynomial of degree d, find the roots of f within ϵ.

Given our assumptions, we can model any polynomial-solving algorithm with a tree. (For the present paper, we exclude any programs with loops in them.) More formally, we adopt the convention of Manna's Flowchart Programs [4].

Date: Summer 2017.

Definition 1.1. An **algorithm** is a rooted tree with a root at the top representing the input and leaves at the bottom representing the output. There are two types of internal nodes:

Computation nodes,  , which output their input modified by a rational function.

Decision nodes,  , at which the program either goes left or right depending on whether some inequality is true or false.

We call these algorithms a "computation tree."

Definition 1.2. The **topological complexity** of an algorithm is the number of decision nodes (intuitively, topologically speaking, computation nodes don't affect the structure of the tree.)

The main theorem of Smale [5], which we will prove in this paper is:

Theorem 1.3 (Smale)**.** *For all* $0 < \epsilon < \epsilon(d)$*, the topological complexity of Poly(d) is (strictly) greater than* $(\log_2(d))^{2/3} - 1$.

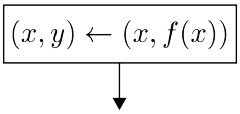
2. Algorithmic processes and nodes

Essentially, an algorithm takes input of a certain type (from a specific space, in our case a subset of the complex monic polynomials), then does computation, and stores intermediary variables in "memory" (also represented as a specific space), and then returns output (another space, in our case, a d-tuple of complex numbers).

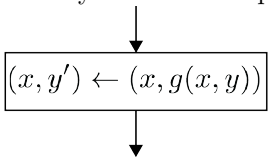
It is important to formalize this notion. The input domain is denoted $\mathcal{I}$, the output domain is denoted $\mathcal{O}$, and the program (computation) domain is denoted $\mathcal{P}$.

There are four types of internal nodes in an algorithm: start, compute, decision, and end.

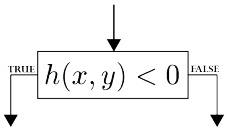
Start nodes are defined by a rational map, $f : \mathcal{I} \to \mathcal{P}$ and perform the first computation on the input, x:



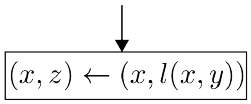
Computation nodes are defined by a rational map, $g : \mathcal{I} \times \mathcal{P} \to \mathcal{P}$:



Each branch (decision node) is a rational map, $h : \mathcal{I} \times \mathcal{P} \to \mathcal{P}$, and upon encountering the branch, the computer makes a decision based on an inequality (could be $<$ or $\leq$):



Finally, end nodes are defined by a rational map, $l : \mathcal{I} \times \mathcal{P} \to \mathcal{O}$, and perform the final computation:



Each input, $x \in \mathcal{I}$ defines a specific path down the tree.

3. From computation theory to algebraic topology

This section is based on a spectacular result by Smale [5], which establishes a connection between computation theory and algorithmic complexity and the covering space theory, so the rest of the proof can proceed by algebraic topology.

The program will not use all the inputs in $\mathcal{I}$ (in our case, we want to only consider polynomials with distinct roots). The set of *usable inputs* is a semialgebraic set $Y \subset \mathcal{I}$. Similarly, the set of *usable outputs* is the set of potential outputs, when the program is given elements of Y. Formally, the algorithm returns an element of $Y \times \mathcal{O}$, so the usable outputs is the set $X \subset Y \times \mathcal{O}$ such that $X \to Y$ is the restriction of the projection $Y \times \mathcal{O} \to Y$ where the projection is required to be surjective. Additionally, we require that if $x \in Y$, then division by zero is never encountered in the algorithm.

Let $f : X \to Y$ be continuous.

Now, we specify $\mathcal{I}$, $\mathcal{P}$, $\mathcal{O}$, X, and Y for Poly(d).

The input space, $\mathcal{I}$ is $\mathscr{P}_d \subset \mathbb{C}[t]$, the space of complex monic polynomials of degree d. $p \in \mathscr{P}_d$ can be written as $p = t^d + a_{d-1}t^{d-1} + \cdots + a_1 t + a_0$, so we can identify $\mathscr{P}_d$ with $\mathbb{C}^d$.

The program space is just $\mathbb{C}^n$ with n sufficiently large.

The program is tasked with approximating the roots of a polynomial, so, naturally, the output space $\mathcal{O}$ is $\mathbb{C}^d$, and is populated by d-tuples of the space of polynomials.

Since restriction of the input space does not increase the complexity, we consider $Y \subset \mathcal{I}$ to be the collection of polynomials with distinct roots, $X \subset Y \times \mathcal{O}$ to be the corresponding output (roots of the polynomials of Y to within ϵ), and $f : X \to Y$ to be the restriction of the projection map $Y \times \mathcal{O} \to Y$.

Let the function $\pi : \mathbb{C}^d \to \mathscr{P}_d$ be the function which maps roots $(\gamma_1, \cdots, \gamma_d)$ to the monic polynomial which vanishes on the γ_i. That is,

$$\pi(\gamma_1, \cdots, \gamma_d) = (t - \gamma_1) \cdots (t - \gamma_d)$$

And the coefficients of π in the natural basis of $\mathscr{P}_d$ are given by the elementary symmetric polynomials in the roots.

Definition 3.1. The **covering number** of $f : X \to Y$ is the smallest k such that there exist open sets $\mathcal{U}_1, \cdots, \mathcal{U}_k \subset Y$ and continuous maps $g_i : \mathcal{U}_i \to X$ such that $f \circ g = \mathrm{Id}_{\mathcal{U}_i}$

Remark 3.2. π is an $d!$-fold cover of $\mathscr{P}_d$ because $\pi(\gamma_1, \cdots, \gamma_d) = (\gamma_{\sigma(1)}, \cdots, \gamma_{\sigma(n)})$ where $\sigma \in \mathfrak{S}_d$, the symmetric group.

We would like to only consider polynomials with distinct roots. Define

$$\Delta := \{(\gamma_1, \cdots, \gamma_d) : \exists\, i \neq j \text{ such that } \gamma_i = \gamma_j\}$$

Then, define $\Sigma := \pi(\Delta)$. Σ consists of the polynomials with repeated roots, and is an algebraic variety (where the discriminant is zero).

Clearly π restricts $\mathbb{C}^d \setminus \Delta \to \mathscr{P}_d \setminus \Sigma$.

For the mechanics of the next theorem, we will need to change the usable inputs and outputs. We pick them very restrictively, and then inverse deformation retract onto X and Y as defined above. For now, the space of usable inputs will be defined as

$$B_K := \{f \in \mathscr{P}_d : |a_i| \leq K, i = 0, 1, \cdots, d-1, t^d + a_{d-1}t^{d-1} + \cdots + a_1 t + a_0\}$$

with $K = K(d)$ sufficiently large so that if f has all of its roots in the unit disk, then $f \in B_K$. Then, the set of acceptable outputs, X is just the set of possible outputs from B_K (the d-tuples which fall within ϵ of the roots of $f \in B_K$).

Now, we prove Smale's connection between computation theory and algebraic topology:

Theorem 3.3 (Smale)**.** *The covering number of the restriction $\pi : \mathbb{C}^d \setminus \Delta \to \mathscr{P}_d \setminus \Sigma$ is less than or equal to the number of branches of Poly(d) for all $\epsilon < \epsilon(d)$ described in the proof.*

Proof. Since any algorithm for Poly(d) can be described as a computation tree, we can number its leaves $i = 1, \cdots, k$. Then, define $V_i \subset B_K$ as the set of polynomials which arrive at leaf i. So, since all polynomials from B_K arrive at one of the leaves, and no polynomial arrives at multiple leaves, the following statements are true: $B_K = \cup_{i=1}^k V_i$ and $V_i \cap V_j = \emptyset$ if $i \neq j$.

Additionally, since the algorithm computes the roots for f, there is a very natural $\varphi_i : V_i \to \mathbb{C}^d$ where $\varphi_i(f) = (z_1, \cdots, z_d)$ such that $|z_i - \gamma_i| < \epsilon$. V_i is closed in B_K (it is a semi-algebraic set). By the Tietze Extension Theorem, we can extend $\varphi_i : \mathcal{U}_i \to B_K$ where $\varphi_i(f) = (z_1, \cdots, z_d)$ such that $|z_i - \gamma_i| < \epsilon$ still.

Now that we have a nice covering of B_K, we would like to "blow this covering up" so it covers $\mathscr{P}_d \setminus \Sigma$. For this, we will need to inverse deformation retract from B_K to the entire space.

Definition 3.4 (Deformation retraction)**.** A subspace $A \subset B$ is a deformation retraction of B if there exists some homotopy $h : I \times B \to B$ such that $h(0, \cdot) = \mathrm{Id}_B$ and $h(1, B) \subset A$ and $h(1, \cdot)$ restricted to A is the identity on A. That is, $h(1, a) = a$ for all $a \in A$.

We now need some Lemmas as technical tools:

Lemma 3.5. *Let A be a closed, compact subspace of a space B such that (B,A) can be triangulated (it is homeomorphic to a simplicial complex and a subcomplex respectively). Then, there exists a neighborhood of A such that $B \setminus N$ is a deformation retract of A.*

Then, let S be the unit sphere in $\mathbb{C}^d$ under the standard Hermitian inner product.

Lemma 3.6. *$(\pi(S), \Sigma \cap \pi(S))$ can be triangulated.*

Lemma 3.7. *$\pi(S) \setminus \Sigma \cap \pi(S)$ is a deformation retract of $\mathscr{P}_d \setminus \Sigma$.*

Proof of Lemma. We can just write down the deformation retraction. $h(t, x) = (1-t)x + t\frac{x}{\|x\|}$. (Note: technically h acts on a vector in $\mathbb{C}^d$, but since it's invariant under the symmetric group, we can say that it acts on the polynomial space. That is, we can use h to bring all of the roots of the polynomials into the unit sphere.) ∎

It follows that:

Lemma 3.8. *There exists a neighborhood, N, of $\Sigma \cap \pi(S)$ such that $\pi(S) \setminus N$ is a deformation retract of $\mathscr{P}_d \setminus \Sigma$.*

As aforementioned, we're going to use this deformation retraction to expand our covering for B_K to the covering for the entire space.

Let $h : \mathscr{P}_d \setminus \Sigma \times I \to \mathscr{P}_d \setminus \Sigma$ be the deformation retraction from Lemma 3.8. That is, $h(\mathscr{P}_d \setminus \Sigma, 1) \subset \pi(S) \setminus N$. We want to show that when we invert this retraction on the covering of B_K, we can uniquely determine a polynomial in $\mathscr{P}_d \setminus \Sigma$ with the appropriate roots. So, choose $\mu(d)$ such that if $f \in \pi(S) \setminus N$, the roots of f satisfy $|\gamma_i - \gamma_j| > \mu(d)$ for $i \neq j$.

Then, choose $P_i = \mathcal{U}_i \cap \pi(S) \setminus N$. Notice that the P_i cover $\pi(S) \setminus N$.

Fix $\epsilon < \frac{\mu(d)}{2}$. This (slightly technical) work with roots and inequalities has been so we can uniquely determine the polynomial and nicely define the g_i in Definition 3.1. We can define $\psi_i(f)$ as follows (for $f \in P_i$):

$$f \xrightarrow{\varphi_i(f)} (z_1, \cdots, z_d) \rightsquigarrow (\gamma_1, \cdots, \gamma_d)$$

Note that the determination of the roots above (denoted by $\rightsquigarrow$) is possible only because each z_i has a closest root of f defined unambiguously. Therefore, $\psi_i : P_i \to \mathbb{C}^d \setminus \Delta$ is continuous and the P_i cover $\pi(S) \setminus N$.

Finally, we define $Q_i := h^{-1}(P_i, 1)$ and using the covering homotopy property, we extend $\psi_i : Q_i \to \mathbb{C}^d \setminus \Delta$.

■

Remark 3.9. Clearly, Theorem 3.3 can be generalized for many other situations.

4. The cup length as a bound on the covering number

We assume familiarity with the notions of cohomology and cohomology ring. We denote the cohomology ring of a topological space, X, as $H^*(X, R)$. We often omit R, and just write the cohomology ring as $H^*(X)$. In these cases, R is irrelevant, and can just be taken to be $\mathbb{Z}$.

$H^*(X, R)$ comes equipped with a cup product. We write $\smile : H^*(X, R) \to H^*(X, R)$ to be this map.

Definition 4.1 (Cup length)**.** The **cup length** of a cohomology ring, $H^*(X; R)$ is the largest k such that there exist $\eta_1, \cdots, \eta_k \in H^*(X; R)$ with $\eta_1 \smile \eta_2 \smile \cdots \smile \eta_k \neq 0$.

In fact, we can bound the covering number of a map by the cup length of the kernel of the induced map on the cohomology.

Lemma 4.2. *If $f : X \to Y$ is continuous, then the covering number of f is strictly greater than the cup length of* $\ker(f^*)$ *where $f^* : H^*(Y) \to H^*(X)$ is the naturally induced map.*

Proof. Suppose by contradiction that there exists some covering, $V_1, \cdots, V_k$ of Y and maps $\sigma_i : V_i \to X$ with $f(\sigma_i(y)) = y$ and, the cup length is also k. So, there exist $\eta_1, \cdots, \eta_k \in \ker(f^*)$ such that $\eta_1 \smile \cdots \smile \eta_k \neq 0$.

Then, the below sequence is exact, and the triangle commutes:

$$\cdots \longrightarrow H^*(Y, V_i) \longrightarrow H^*(Y) \longrightarrow H^*(V_i) \longrightarrow \cdots$$
$$\searrow^{f^*} \quad \uparrow{\scriptstyle \sigma_i^*}$$
$$H^*(X)$$

Then, we look at preimages of the η_i in $H^*(Y, V_i)$. These exist since the sequence is exact; and, the image of η_i in $H^*(V_i)$ is zero because the triangle commutes. We then choose $v_i \in H^*(Y, V_i)$ such that $v_i \mapsto \eta_i$ under the first map.

Then, $v_1 \smile v_2 \smile \cdots \smile v_k \in H^*(Y, V_1 \cup V_2 \cup \cdots \cup V_k) = H^*(Y, Y) = 0$. But, this is a contradiction, since $v_1 \smile v_2 \smile \cdots \smile v_k \mapsto \eta_1 \smile \eta_2 \smile \cdots \smile \eta_k \neq 0$. ■

5. Computing the kernel of π^*

We need another lemma to be able to apply Lemma 4.2 to our problem.

Lemma 5.1. *Let $\pi : \mathbb{C}^d \setminus \Delta \to \mathscr{P}_d \setminus \Sigma$ as previously defined. Then, the induced map in cohomology (coefficients in $\mathbb{Z}/2\mathbb{Z}$):*

$$\pi^* : H^i(\mathscr{P}_d \setminus \Sigma, \mathbb{Z}/2\mathbb{Z}) \to H^i(\mathbb{C}^d \setminus \Delta, \mathbb{Z}/2\mathbb{Z})$$

is trivial for $i > 0$.

Proof. We need to use some groups, their classifying spaces, and other constructions from them. Let $O(n)$ be the orthogonal group, $BO(n)$ be the corresponding classifying space. Let $\mathfrak{S}_n$ be the symmetric group, and $B\mathfrak{S}_n = K(\mathfrak{S}_n, 1)$ be an Eilenberg-MacLane space. Finally, let $U_n \to B\mathfrak{S}_n$ be the universal cover.

It is a well-known result that $\mathscr{P}_d \setminus \Sigma$ is an Eilenberg-MacLane space, $K(\pi_1(\mathscr{P}_d \setminus \Sigma), 1)$. It is important to note that $\pi_1(\mathscr{P}_d \setminus \Sigma) = \mathrm{Bd}(d)$, the braid group in d strands. As per Remark 3.2, π is a covering map with group $\mathfrak{S}_d$. Then, there's a natural map:

$$\pi_1(\mathscr{P}_d \setminus \Sigma) = \mathrm{Bd}(d) \to \mathfrak{S}_d$$

This map is realized by taking the starting and ending positions of the strands. There's also a natural $\mathfrak{S}_d$ action on a basis, $e_1, \cdots, e_d \mapsto e_{\sigma(1)}, \cdots, e_{\sigma(d)}$ which realizes a map

$$\mathfrak{S}_d \to O(d)$$

Ring homomorphisms in cohomology over $\mathbb{Z}/2\mathbb{Z}$ are induced by the group homomorphisms, so the above maps give rise to a map on the cohomologies:

$$H^*(\mathscr{P}_d \setminus \Sigma) \leftarrow H^*(B\mathfrak{S}_d) \leftarrow H^*(BO(d))$$

According to Fuchs [2], the composition, $H^*(BO(d), \mathbb{Z}/2\mathbb{Z}) \to H^*(\mathscr{P}_d \setminus \Sigma, \mathbb{Z}/2\mathbb{Z})$ is surjective. Moreover, the composition

$$\pi_1(\mathbb{C}^d \setminus \Delta) \to \pi_1(\mathscr{P}_d \setminus \Sigma) \to \pi_1(B\mathfrak{S}_d) \simeq \mathfrak{S}_d$$

is trivial. So, by covering space theory, it can be lifted to the universal cover, as defined earlier. That is, there exists φ such that

$$\begin{array}{ccc} \mathbb{C}^d \setminus \Delta & \xrightarrow{\varphi} & U_d \\ \downarrow{\scriptstyle \pi} & & \downarrow \\ \mathscr{P}_d \setminus \Sigma & \longrightarrow & B\mathfrak{S}_d \end{array}$$

commutes. Then, since all the cohomologies of U_d are isomorphic to $H^0(U_d, \mathbb{Z}/2\mathbb{Z})$, we know that the composition $H^i(B\mathfrak{S}_d, \mathbb{Z}/2\mathbb{Z}) \to H^i(\mathbb{C}^d \setminus \Delta, \mathbb{Z}/2\mathbb{Z})$ is trivial for $i > 0$. Then, the Lemma follows using the result from Fuchs. ■

Define

$$H^{>0}(\mathscr{P}_d \setminus \Sigma, \mathbb{Z}/2\mathbb{Z}) = \sum_{i=1}^{2d} H^i(\mathscr{P}_d \setminus \Sigma, \mathbb{Z}/2\mathbb{Z})$$

Then, the last result we need is the core result of Fuchs.

Lemma 5.2. *The cup length of $H^{>0}(\mathscr{P}_d \setminus \Sigma, \mathbb{Z}/2\mathbb{Z})$ is greater than $(\log_2(d))^{2/3}$.*

Proof. Fuchs [2] proved that the generators of $H^{>0}(\mathscr{P}_d \setminus \Sigma, \mathbb{Z}/2\mathbb{Z})$ are $g_{m,k}$ with $k = 0, 1, 2, \cdots$ and $m = 1, 2, 3, \cdots$ with the degree of $g_{m,k}$ equal to $2^k \cdot 2^{m-1}$, and the following relations: 1) $g_{m,k}^2 = 0$ and otherwise, 2) $g_{m_1,k_1} \cdot g_{m_2,k_2} \cdots g_{m_n,k_n} = 0$ only if $2^{m_1+m_2+\cdots+m_n+k_1+k_2+\cdots+k_n} > d$.

Taking the $\log_2$ of both sides shows that we are looking for pairs (m_i, k_i) with $i = 1, \cdots, n$ such that n is as large as possible, and $\sum m_i + \sum k_i \leq \log_2(d)$. To count these, we can first count the number of pairs (m_i, k_i) that satisfy $m_i + k_i \leq N$ for some fixed N. A straightforward counting, shows that there are $\sum_1^N i = \frac{N(N+1)}{2}$ of these pairs.

It is also straightforward that $\sum m_i + \sum k_i \leq \log_2(d)$ will be satisfied provided

$$\sum_1^M i^2 = \frac{M(M+1)(2M+1)}{6} \leq \log_2(d)$$

Finally, we check that $n = (\log_2(d))^{2/3}$ satisfies these conditions. Thus, the proof is complete. ■

6. Conclusion

The proof of Theorem 1.3 now follows:

Topological complexity of Poly(d) +1	
= Number of leaves in Poly(d)	Definition
$\geq$ Covering number of $\pi : \mathbb{C}^d \setminus \Delta \to \mathscr{P}_d \setminus \Sigma$	Theorem 3.3
$>$ Cup length of $\ker(\pi^*)$	Lemma 4.2
= Cup length of $H^{>0}(\mathscr{P}_d \setminus \Sigma, \mathbb{Z}/2\mathbb{Z})$	Lemma 5.1
$\geq (\log_2(d))^{2/3}$	Lemma 5.2

7. Low-complexity algorithms via Newton's Method on the complex plane

Theorem 1.3 gives rise to an important problem. One wishes to find algorithms for computing the roots of polynomials with minimal topological complexity. Since a lower bound is given on the problem, it serves as the target complexity for an algorithm.

Moreover, finding such an algorithm gives an upper bound for the algorithm. We presently explore two methods of approximating roots. The first, discussed in this section, approximates roots of a polynomial via an extension of Newton's Method to the complex plane.

7.1. **Quadratic polynomials.** Given $f(t) = t^2 + a_1 t + a_0$, one can find its roots by applying the quadratic formula.

$$t = \frac{-a_1 \pm \sqrt{\Omega}}{2}$$

In the above formula, Ω is the discriminant, $a_1^2 - 4a_0$. So, the problem essentially reduces to finding the square root of a complex number. This can be done if we restrict $\sqrt{\cdot} : \mathbb{R} \to \mathbb{R}$ by applying Newton's method. We presently extend Newton's method to the complex numbers with a computation-theoretic approach.

Given a complex number S, we can approximate the solution to the equation $x^2 - S = 0$ by starting with a "seed", x_0, and then defining the sequence:

$$x_{n+1} = x_n - \frac{x_n^2 - S}{2x_n} = x_n - \frac{(x^2 - S)|_{x=x_n}}{\frac{\mathrm{d}}{\mathrm{d}\,x}(x^2 - S)|_{x=x_n}}$$

If the seed is chosen correctly, this sequence converges to $\sqrt{S}$. So, the only branching occurs in choosing the seed.

We created a program in Python to illustrate the duration of convergence when we start with $x_0 = 1$, and graph S on the complex plane:

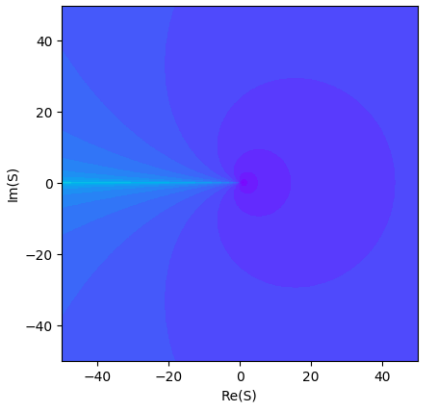


FIGURE 1. The duration for convergence of $x_n \to \sqrt{S}$ within a radius of 0.1, seeded by the number 1. Purple indicates rapid convergence, and light blue indicates divergence.

Definition 7.1. A **fractal diagram for $t^d - S$ seeded by k to threshold r** is a diagram (as above) showing the duration of convergence for $x_n \to \sqrt[d]{S}$ within a radius of r, seeded by the number k where purple indicates rapid convergence and light blue indicates divergence. The sequence x_n is given by Newton's method extended to the complex plane as defined at the beginning of this section.

There is clearly a problem with using this seed for $\mathrm{Re}(S) < 0$, especially on the negative x-axis. So, we can compute the same diagram for $x_0 = i$.

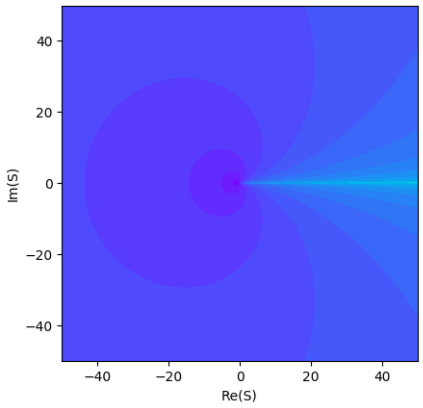


FIGURE 2. Fractal diagram for $t^2 - S$ seeded by i to threshold 0.1.

We have therefore identified the branch in an efficient algorithm for Poly(d):
$f(t) = t^2 + a_1 t + a_0$
Define $\Omega = a_1^2 - 4a_0$.
If $\mathrm{Re}(\Omega) < 0$: Perform Newton's Method to compute $\sqrt{\Omega}$ with seed i.
Otherwise: Perform Newton's Method to compute $\sqrt{\Omega}$ with seed 1.
Return $t = \frac{1}{2}(-a_1 + \sqrt{\Omega}), \frac{1}{2}(-a_1 - \sqrt{\Omega})$.

This algorithm, has topological complexity 1. Indeed, $(\log_2(2))^{2/3} - 1 = 0$, so the number of branches is strictly greater than 0, and this algorithm is therefore minimal.

Remark 7.2. This algorithm does not violate Smale's no-loop condition because we can guarantee that for $S \in B_K$, Newton's method terminates in less than N steps for some large N.

7.2. **Cubic polynomials.** We similarly make use of the cubic formula, which requires computing the square root and the cube root. Specifically, the formula requires computing one square root, and two cube roots.

We can similarly compute the cube root of a number, S, by computing the sequence:

$$x_{n+1} = x_n - \frac{x_n^3 - S}{3x_n^2}$$

If x_0 is chosen properly, this sequence similarly converges to $\sqrt[3]{S}$. Using the same program, we created a similar graph as shown for the $\sqrt{\cdot}$ case.

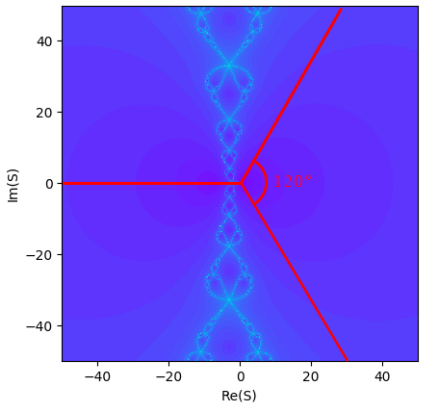


FIGURE 3. Fractal diagram for $t^3 - S$ seeded by 1 to threshold 0.1.

As noted in the diagram, the algorithm, seeded by 1, converges rapidly for S in a specific third of the coordinate plane. We can modify the seed so that the algorithm converges in different sections.

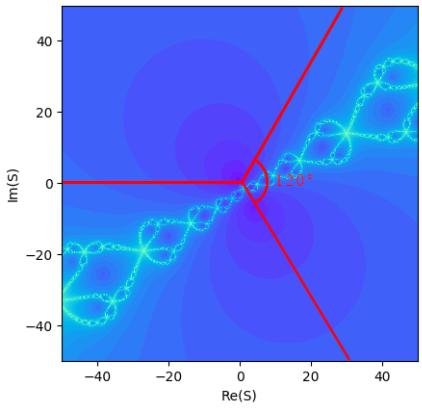


FIGURE 4. Fractal diagram for $t^3 - S$ seeded by $e^{\frac{2\pi}{9}i} \approx 0.77 + 0.64i$ to threshold 0.1.

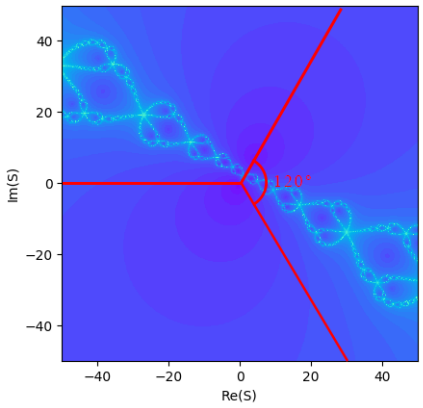


FIGURE 5. Fractal diagram for $t^3 - S$ seeded by $e^{\frac{4\pi}{9}i} \approx 0.17 + 0.98i$ to threshold 0.1.

Therefore, we have cut the complex plane into three pieces, and identified a seed for each such that the algorithm converges rapidly in each when seeded appropriately. So, the corresponding algorithmic tree contains two branches.

Altogether, finding the roots of a general cubic equation requires at most five branches.

7.3. **Quartic polynomials.** We can continue the approach of using closed form expressions for the roots of degree d polynomials only up to $d = 4$. In fact, for the quartic case, we must add additional branches, because there is a risk of dividing by zero, which does not occur in the other expressions.

Let $f(t) = t^4 + a_3t^3 + a_2t^2 + a_1t + a_0$. We need to first perform several computations:

$$\begin{aligned} p &= \frac{8a_2 - 3a_3^2}{8} \\ q &= \frac{a_3^3 - 4a_3a_2 + 8a_1}{8} \\ \Omega_0 &= a_2^3 - 3a_3a_1 + 12a_0 \\ \Omega_1 &= 2a_2^3 - 9a_3a_2a_1 + 27a_3^2a_0 + 27a_1^2 - 72a_2a_0 \end{aligned}$$

Thus far we have used no branches. Then define

$$Q = \sqrt[3]{\frac{\Omega_1 + \sqrt{\Omega_1^2 - 4\Omega_0^3}}{2}}$$

using 3 branches. We would then like to define

$$S = \frac{1}{2}\sqrt{-\frac{2}{3}p + \frac{1}{3}\left(Q + \frac{\Omega_0}{Q}\right)}$$

and ideally use just one more branch. But, we need to handle the case that $Q = 0$ with a separate branch. If $Q = 0$, though, at least three of the roots are equal, and

the roots are rational functions in the coefficients. Thus far, then, we have used five branches.

If $Q \neq 0$, then

$$t_{1,2} = -\frac{b}{4} - S \pm \sqrt{-4S^2 - 2p + \frac{q}{S}} \tag{7.3}$$

$$t_{3,4} = -\frac{b}{4} - S \pm \sqrt{-4S^2 - 2p - \frac{q}{S}} \tag{7.4}$$

using two more branches. Altogether, solving a quartic requires at least seven branches.

7.4. $t^d - S$**.** The approach to solving general quadratic, cubic, and quartic equations was enabled entirely by computing square and cube roots. In general,

Theorem 7.5. *The class of degree d, complex, monic polynomials which can be expressed as $t^d - S$ can be solved in d branches.*

While a rigorous proof of this theorem is not given, we hope to provide significant insight into this result.

Firstly, the fractal diagram for $t^d - S$ seeded by k to threshold r contains $d-1$ "branches" for sufficiently small r. A "branch" on a fractal diagram refers to the visual branches evident in the diagram starting at a point near the origin and extending to infinity, growing in size. So, for a given seed, we can only guarantee convergence for polynomials in one of d sections of the plane. This breakdown is done explicitly in Section 7.2. Another example is given below.

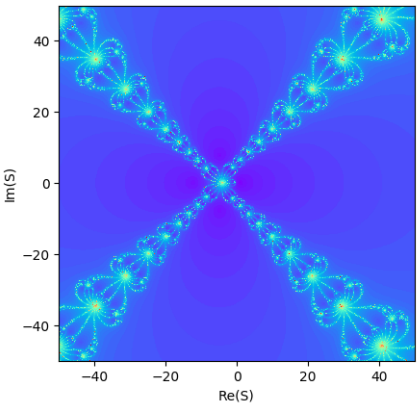


FIGURE 6. Fractal diagram for $t^5 - S$ seeded by 1 to threshold 0.1.

Secondly, the fractal diagram for $t^d - S$ seeded by $e^{\frac{2\pi}{d^2}ik}$ $(k = 0, 1, 2, \cdots, d-1)$ is exactly the fractal diagram for $t^d - S$ seeded by 1 rotated clockwise by $\frac{2\pi k}{d}$ radians.

These two observations give a concrete algorithm to compute the roots of $t^d - S$ in d branches with the branching based on the location of S in the complex plane.

Remark 7.6. Technically, the algorithm only gives one root of $t^d - S$, but the other roots are given by the roots of unity times the root the algorithm gives.

8. Low-complexity algorithms via Power Iteration

Given $f(t) \in B_K$ with $\deg(f) = d$ in coefficients $a_{d-1}, \cdots, a_0$, we can construct

$$F = \begin{pmatrix} 0 & 0 & \cdots & 0 & -a_0 \\ 1 & 0 & \cdots & 0 & -a_1 \\ 0 & 1 & \cdots & 0 & -a_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & -a_{d-1} \end{pmatrix}$$

the companion matrix of $f(t)$. The characteristic polynomial of F is f. Therefore, approximating the roots of the polynomial can be done by approximating the eigenvalues of F. One such algorithm to do this is called power iteration (see Bindel [1] for a more extensive definition).

The power iteration algorithm (almost always) returns an eigenvalue of a given $(n \times n)$ matrix A. It proceeds as follows:

(1) Choose a "seed" vector, b_0. In general, this vector is chosen randomly. For our specific case, with F defined as above, the only vector we need to use is $(0, 0, \cdots, 0, 1)$. See Remark 8.2 after the proof for an explanation as to why.

(2) Compute the values of the sequence $b_n = \frac{Ab_{n-1}}{\|Ab_{n-1}\|}$.

"In general", $b_n \to v$ where $Av = \lambda v$. Formally:

Theorem 8.1. *If A is an $n \times n$ matrix, with eigenvalues $|\lambda_1| > |\lambda_2| > |\lambda_3| > \cdots > |\lambda_k| > 0$, then power iteration converges to a vector v with $Av = \lambda_1 v$, unless b_0 is orthogonal to v.*

Proof. We can write A in Jordan canonical form, ordered such that:

$$J = \begin{pmatrix} J_1 & & & \\ & J_2 & & \\ & & \ddots & \\ & & & J_k \end{pmatrix}$$

where J_i is the Jordan block corresponding to λ_i. Written this way, $A = VJV^{-1}$. Then, we can write b_0 in the columns of V. Write $b_0 = \alpha_1 v_1 + \cdots + \alpha_n v_n$.

Assume that $\alpha_1 \neq 0$, so b_0 has a nonzero component in the direction of v_1.

Then, compute:

$$\begin{aligned} b_k &= \frac{A^k b_0}{\|A^k b_0\|} \\ &= \frac{(VJV^{-1})^k b_0}{\|(VJV^{-1})^k b_0\|} \\ &= \frac{VJ^kV^{-1}(\alpha_1 v_1 + \cdots + \alpha_n v_n)}{\|VJ^kV^{-1}(\alpha_1 v_1 + \cdots + \alpha_n v_n)\|} \\ &= \left(\frac{\lambda_1}{|\lambda_1|}\right)^k \frac{\alpha_1}{|\alpha_1|} \frac{v_1 + \frac{1}{\alpha_1} V \left(\frac{1}{\lambda_1} J\right)^k (\alpha_2 e_2 + \cdots + \alpha_n e_n)}{\|v_1 + \frac{1}{\alpha_1} V \left(\frac{1}{\lambda_1} J\right)^k (\alpha_2 e_2 + \cdots + \alpha_n e_n)\|} \end{aligned}$$

And since λ_1 is dominant,

$$(\frac{1}{\lambda_1}J)^k \to \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix}$$

so, written as above, much of the fraction in our final expression for b_k dies off as $k \to \infty$. It follows that $b_k \to \frac{v_1}{\|v_1\|}$. ■

Remark 8.2. If $|\lambda_1| > |\lambda_2| > |\lambda_3| > \cdots > |\lambda_k| > 0$, then power iteration always converges to the eigenvector corresponding to λ_1, unless b_0 is orthogonal to the eigenvector. Computationally, one chooses b_0 randomly to give a high likelihood that b_0 is not orthogonal to the eigenvector.

In a basis, at least one vector is not orthogonal to the eigenvector. So, if we perform power iteration on the entire canonical basis, at least one is guaranteed to converge. For F above, it is clear that the vector $b_0 = (0, 0, \cdots, 0, 1)$ is sufficient to guarantee convergence, because all other basis vectors turn into that vector at some point during the algorithm ($Fe_i = e_{i+1}$ for $i < n$).

Remark 8.3. Power iteration converges geometrically, with ratio $|\frac{\lambda_2}{\lambda_1}|$.

Notice that Remark 8.3 is especially problematic. It means that we cannot guarantee that power iteration converges quickly, which is important, since Poly(d) excludes programs with loops in them.

Earlier in the paper (e.g., Section 7), we have made the notion of "quick" convergence (i.e. no loops in the program) a mathematical statement. We would like to guarantee that there is a universal N such that after N computations, the algorithm will have arrived within ϵ of the correct answer. For power iteration, though, we must make a different statement.

We can conclude from Remark 8.3 that there exists some sufficiently large $N = N(\delta)$ such that for all $1 > \delta > 0$, we can construct a program, based on power iteration which can find all the roots of f with *no* branches, many iterations, and $b_0 = (0, 0, \cdots, 0, 1)$, terminating in N steps, which finds the roots to within ϵ with *probability* greater than δ.

Remark 8.4. The challenge is only to find a single root of f. Then, we can divide f by $t - \lambda$, and repeat the procedure.

The careful reader will notice that above, we exclude the case where $|\lambda_1| = |\lambda_2|$. This case is problematic.

8.1. **Equal-magnitude dominant and subdominant eigenvalues.** If $|\lambda_1| = |\lambda_2|$, but $\lambda_1 \neq \lambda_2$ (i.e. $\lambda_1 = re^{\theta_1 i}$ and $\lambda_2 = re^{\theta_2 i}$ with $\theta_1 \neq \theta_2$), then power iteration does not converge to a single vector. Instead, it eventually approaches the sequence

$$e^{n\theta_1 i} + e^{n\theta_2 i}$$

In this case the goal is to isolate θ_1 and θ_2 from the values of the sequence. We suspect this is difficult, and cannot be done without using many branches.

Acknowledgments. I would like to thank my mentor, Weinan Lin, for teaching me much of the material presented here, and Shmuel Weinberger for giving me this topic and directing me appropriately. I would like to especially thank J. Peter May for organizing the REU at the University of Chicago, which I attended during the summer of 2017.

References

[1] David Bindel. Week 10: Wednesday, Oct 28. CS 6210 Notes.
http://www.cs.cornell.edu/ bindel/class/cs6210-f09/lec26.pdf
[2] D. B. Fuchs. Cohomologies of the braid groups mod 2.
http://www.mathnet.ru/links/349e943ca5398fab16a8902582209686/faa2653.pdf
[3] Allen Hatcher. Algebraic topology. Cambridge University Press.
https://www.math.cornell.edu/ hatcher/AT/AT.pdf
[4] Zohar Manna. Introduction to Mathematical Theory of Computation (Computer Science). McGraw-Hill Education (December 1974).
[5] Steve Smale. On the Topology of Algorithms, I.
http://www.sciencedirect.com/science/article/pii/0885064X87900215