Root-Finding Algorithms — Instructor: Padraic Bartlett

An Introduction to Root-Finding Algorithms

Day 1 — *Mathcamp 2013*

1 Introduction

How do we find the roots of a given function? This, on one hand, is a task we've been studying and working on since grade school. For example, it's hard to get out of a modern algebra[1] class without looking at linear maps like $f(x) = ax + b$, and showing that these functions have their only roots at $x = -\frac{b}{a}$ (assuming that a is nonzero.)

Shortly afterwards, most algebra classes run through the **quadratic formula**, which tells us that the only roots of a polynomial of the form $ax^2 + bx + c$ occur when

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

A few particularly cruel classrooms may force their students to memorize the following formula for the three potentially distinct roots of an arbitrary cubic polynomial $ax^3 + bx^2 + cx + d$:

$$x_k = -\frac{1}{3a}\left(b + u_k C + \frac{\Delta_0}{u_k C}\right),\ k \in \{1, 2, 3\}$$

where

$$u_1 = 1, u_2 = \frac{-1 + i\sqrt{3}}{2}, u_3 = \frac{-1 - i\sqrt{3}}{2}$$

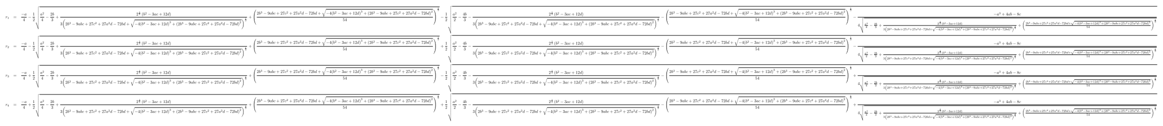
are the three **third roots of unity**[2], and

$$C = \sqrt[3]{\frac{2b^3 - 9abc + 27a^2d + \sqrt{-27a^2(18abcd - 4b^3d + b^2c^2 - 4ac^3 - 27a^2d^2)}}{2}}.$$

[1]Lower-case-a algebra, as opposed to the upper-case-a Algebra with groups and fields and such.

[2]For those of you who haven't seen complex numbers/roots of unity before, don't worry about this: we aren't using these objects in our class at all. But if you want a quick definition: let i denote a symbol with the property that $i^2 = -1$, i.e. i is " the square root of negative 1." Let the complex numbers denote the set $\mathbb{C} = \{x + iy : x, y \in \mathbb{R}.\}$. A ***n*-th root of unity** is a complex number $z = x + iy$ such that z^n is 1. For example, the only second roots of unity are $+1$ and -1. The third roots of unity are the three complex numbers u_1, u_2, u_3 such that when they are cubed, we get 1.

There is even a general formula for the roots of an arbitrary quartic polynomial $ax^4 + bx^3 + cx^2 + dx + e$. It is awful and we do not speak of it. But it does exist! Here's a completely useless picture.



Trust me, that picture isn't any more useful at a resolution where you can read it.

Based on these results, you might expect that for quintics and higher-order polynomials, we can do the same thing and just come up with progressively-more-awful formulas, and that **root-finding algorithms** are just fancy ways of deriving these objects.

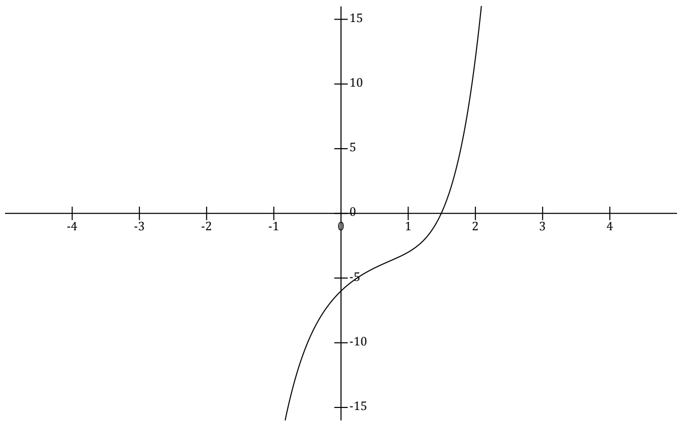
This, surprisingly, is not the case.

Theorem 1 *(Abel-Ruffini) Take any n greater than or equal to five. There is no general algebraic solution – i.e. solution expressed in terms of the coefficients of the polynomial, using only the operations of addition, subtraction, multiplication, division, and radicals – that can give us the roots of an arbitrary degree-n polynomial.*

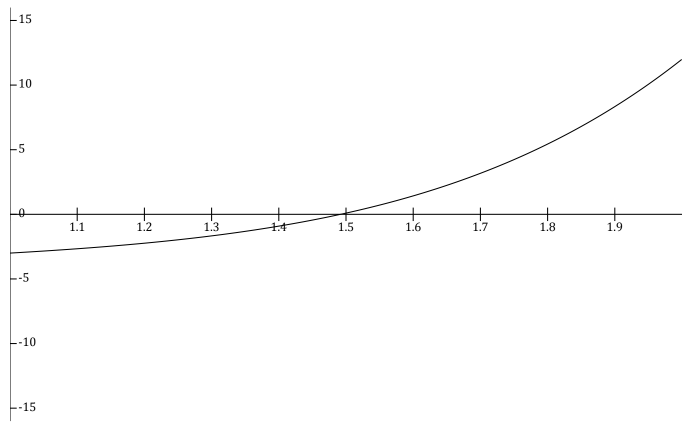
The proof of this theorem is one of the most beautiful results in mathematics; take some classes in Galois theory to learn more! However, this is not the focus of our class.

Instead, our class is motivated by the consequences of this theorem. Suppose we have an arbitrary quintic polynomial of the form $ax^5 + bx^4 + cx^3 + dx^2 + 3x + f$. On one hand, we know that this function must have a root if a is nonzero. (Why?) On the other hand, Abel-Ruffini says that there is no well-behaved formula with which we can express this root: i.e. there is no "quintic formula" which we can throw the constants a, b, c, d, e into and get an expression for the root.

Does Abel-Ruffini mean that we just give up all hope? Does it mean that the roots of something like $x^5 - 2x^4 + 3x^3 - 4x^2 + 5x - 6$ are unknowable, and we should only study systems that use 4th-degree or lower polynomials? The answer, as you probably know, is **of course not**! We might not have a closed and well-behaved formula for the roots of these polynomials, but we certainly can make good guesses at them. For example, if we just graph the above polynomial (to be precise, if we just plug in a finite but large number of points into our polynomial and place these points on a graph,) we can see that it certainly has a root somewhere between 1 and 2.



If we plot more points, we can improve our guess for where this root occurs to somewhere quite close to 1.5.



By repeating this process — graph a number of points, zoom in to a place where the function appears to cross the x-axis, graph more points — we can get increasingly precise approximations to our root. However, this method seems pretty sloppy: we have to graph tons of points at each step to get an idea for where our root is, which could be computationally rather expensive for large-degree polynomials.

In this class, we are going to examine algorithms like the one above: systems that take in polynomials and through a defined sequence of operations find an approximation to the roots of that polynomial. We're going to study a lot of questions with respect to these algorithms: how **fast** do they run, are there inputs for which they **fail**, etc. At first, however, let's start simple: let's work with just a quintic polynomial, and let's suppose that we want to just find one root (not necessarily all of the roots.) What's an easily-described, guaranteed-to-work method that will find a root?

2 Starting Simple: The Bisection Method

We want to find a root of the polynomial

$$ax^5 + bx^4 + cx^3 + dx^2 + ex + f,$$

for arbitrary constants a, b, c, d, e, f with $a \neq 0$. To simplify things, we can assume that $a > 0$ by multiplying through by ± 1; because this is just multiplying our function by a nonzero scalar, this doesn't change where any of the roots of our polynomial lie. So: to find this root, it might be useful to review why we even believe a root exists in the first place! There are a number of arguments you can use to show this polynomial has a root, but perhaps the simplest one is just the definition of continuity.

Specifically: we know that as x goes to infinity in the positive direction, we have

$$\lim_{x \to +\infty} ax^5 + bx^4 + cx^3 + dx^2 + ex + f = +\infty,$$

because for sufficiently large values of x, the x^5 term dominates the above expression. Consequently, we know that for sufficiently large values of x, this polynomial is positive.

(In particular, any value of x greater than both 1 and $\frac{|b|+|c|+|d|+|e|+|f|}{a}$ works. If you don't see why this is true, prove it!)

Similarly, we know that as x goes to infinity in the negative direction, we have

$$\lim_{x \to -\infty} ax^5 + bx^4 + cx^3 + dx^2 + ex + f = -\infty,$$

again because for sufficiently large values of x, the x^5 term dominates the above expression. Again, this tells us in particular that this polynomial takes on negative values for sufficiently large and negative choices of x.

Therefore, because our function takes on a negative value at some input, a positive value at some other input, and is continuous, there must be some value in between where it is zero – i.e. some value where it contains a root!

In particular, if we set $c = \max\left\{1, \frac{|b|+|c|+|d|+|e|+|f|}{a}\right\}$, we've actually shown that our polynomial has a root somewhere in the interval $(-c, c)$. So, in a sense, we can regard this as a particularly bad guess at where a root for our polynomial is! I.e. we have just shown that 0 is approximately a root for our polynomial, with error at most c.

This, however, is a pretty large amount of error. In practice, we want to be able to do better: we want to be able to specify some set amount of error up front, like say 10^{-5}, and have our algorithm find an approximation that's no further from a root than our error bounds. How can we do this?

Well: think back to your earliest math classes. Basically, what we've done so far is classic guess-and-check heuristics that you played around with in grad school: we plugged in something to a function and got an answer that was "too big," and did it again and got an answer that was "too small." So, if we were back in grade school, the next appropriate step would probably be to take something "in the middle," and plug that in to our function as well!

Do this! In particular, we started with the interval $(-c, c)$, and knew that our polynomial was negative at $-c$ and positive at c. So, just take the middle point of this interval — i.e. 0 — and plug that into our polynomial! We will again get some value. If it's 0, then yay we guessed a root, but this is pretty unlikely: we probably got some other value, that's either positive or negative.

If so: return to the guess and check process! In particular, we have that our function is negative at $-c$, something at 0, and positive at c. So between either $-c$ and 0 or 0 and c, our function changes signs: so it has to have a root on one of those two intervals! In other words, we've just improved our error range by a factor of 2: our root is now approximately one of $\pm c/2$, with an error of $c/2$.

If we want to improve by another factor of 2, we can do this again! — just take our interval, pick the midpoint, evaluate the function at that midpoint, and either (1) be really lucky and have that be a root or (2) take the new interval on which our function changes sign. Repeating this k times gives us an error of $\frac{c}{2^{k-1}}$, which by taking large values of k we can make as small as we want!

So, we've just demonstrated an algorithm that works on **any** quintic polynomial — or indeed any continuous function for which we have one point at which it's negative and another at which it's positive! Furthermore, we've studied its efficiency at the same time: we know that it needs k steps to get an error of at most $\frac{1}{2^{k-1}} \cdot$ (interval length).

To give us practice with writing down algorithms in a careful, clearly-stated fashion, we formalize this algorithm below:

Root-Finding Algorithm 1: The Bisection Method

Input: A continuous function $f(x)$, along with an interval $[a, b]$ such that $f(x)$ takes on different signs on the endpoints of this interval, and an error tolerance ϵ.

1. If $b - a < 2\epsilon$, halt. $\frac{b+a}{2}$ is an approximation to a root of $f(x)$ that is within ϵ of some actual root.
2. Using the interval $[a, b]$, define $c = \frac{a+b}{2}$.
3. Evaluate our function $f(x)$ at c. Depending on the sign of $f(c)$, choose whichever of the two intervals $[a, c]$ or $[c, b]$ our function changes sign on.
4. Return to step 2.

Output: A value c that differs from a root of $f(x)$ by at most ϵ.

To illustrate this method, we run an example here:

Example. Find an approximation to a root of $f(x) = x^5 - 2x^4 + 3x^3 - 4x^2 + 5x - 6$ with error at most $\frac{1}{10}$, starting on the interval $[-2, 2]$.

Answer. We first note that at $x = -2$, our function is -120, while at $x = 2$ our function is 12. So we can apply the bisection method to this interval!

Using our method, we repeatedly evaluate our function at the midpoints of intervals and subdivide said intervals. We record this process in the table below:

interval $[a, b]$	midpoint of $[a, b]$	function at midpoint
$[-2, 2]$	0	$f(0) = -6.$
$[0, 2]$	1	$f(1) = -3.$
$[1, 2]$	1.5	$f(1.5) \approx .094.$
$[1, 1.5]$	1.25	$f(1.25) \approx -1.97.$
$[1.25, 1.5]$	1.375	$f(1.375) \approx -1.12.$

The interval $[1.375, 1.5]$ has diameter less than twice our error tolerance. Therefore, we know that an approximation to a root of our function is the midpoint of this interval 1.4375.

3 Slightly Stranger: The False Position Method

Our earlier method works well whenever we can apply it: it's guaranteed to work, we can see exactly how fast it is at getting new digits (about one binary digit per iteration,) and the steps involved in its execution are simple and easy to code (we're just evaluating polynomials and comparing/averaging numbers.)

Can we do better, though? Again, for simplicity's sake, let's assume that we're working with a continuous function $f(x)$ and an interval $[a, b]$ such that $f(x)$ adopts different signs on its endpoints. Can we make a method that converges **faster**? Well: in our previous method, the only piece of data we used was the fact that $f(a)$ and $f(b)$ adopted opposite signs. This, in a sense, is limiting ourselves: by plugging in a and b into our function, we actually know the **values** f takes on these endpoints, rather than just the signs of these values. In a very, *very* limited sense, then, we can regard our function as being "more like" the line through these two points $(a, f(a)), (b, f(b))$ than not!

So: let's solve for the line through these two points. If we want it to go from $(a, f(a))$ to $(b, f(b))$, it should have rise over run — i.e. slope — given by $\frac{f(b)-f(a)}{b-a}$. As well, when we plug in b we should get $f(b)$. If this is true, our "best guess" for our function is the line

$$y = \frac{f(b) - f(a)}{b - a}(x - b) + f(b).$$

This line has a root when

$$\begin{aligned} 0 &= \frac{f(b) - f(a)}{b - a}(x - b) + f(b) \\ \Rightarrow x &= b - f(b)\frac{b - a}{f(b) - f(a)} \\ &= \frac{a \cdot f(b) - b \cdot f(a)}{f(b) - f(a)}, \end{aligned}$$

Take this hypothetical root, and call it c. Plug it into our function. If we were incredibly lucky, we landed on a root, and we're done. Much more likely, we've landed on a new point c at which $f(x)$ has some sign, either positive or negative. Just like before, choose whichever of the two intervals $[a, c], [c, b]$ are such that $f(x)$ switches sign on the endpoints of this interval, and repeat our process!

We write this up formally below.

Root-Finding Algorithm 2: The False-Position Method

Input: A continuous function $f(x)$, along with an interval $[a, b]$ such that $f(x)$ takes on different signs on the endpoints of this interval, and an error tolerance ϵ.

1. If $b - a < 2\epsilon$, halt. $\frac{b+a}{2}$ is an approximation to a root of $f(x)$ that is within ϵ of some actual root.
2. Using our interval $[a, b]$, define $c = \frac{a \cdot f(b) - b \cdot f(a)}{f(b) - f(a)}$.
3. Evaluate our function $f(x)$ at c. Depending on the sign of $f(c)$, choose whichever of the two intervals $[a, c]$ or $[c, b]$ our function changes sign on.
4. Return to step 2.

Output: A value c that differs from a root of $f(x)$ by at most ϵ.

Unlike our earlier example, it's not particularly clear whether this works. So: let's test it!

Example. Find an approximation to a root of $f(x) = x^5 - 3x^3 + 5x - 1$ with error at most $\frac{1}{10}$, starting on the interval $[-1, 1]$.

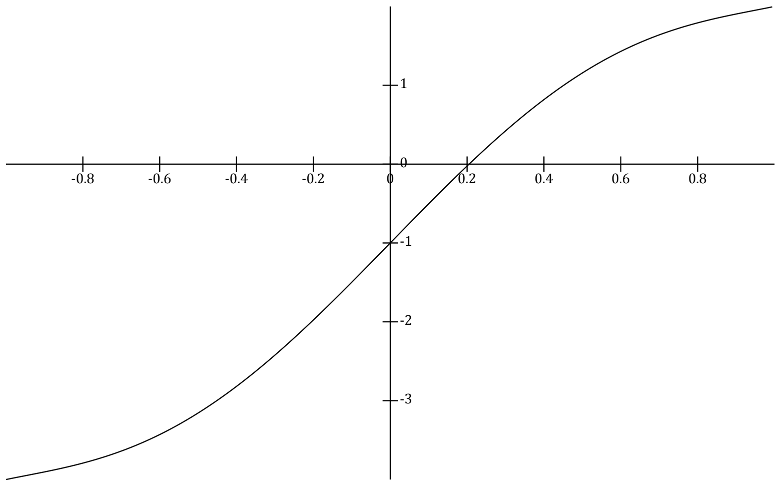
Answer. Again, we first note that at $x = 0$, our function is -6, while at $x = 2$ our function is 12. So we can apply the false position method to this interval!

Using our method, we repeatedly find the root of the line that connects $(a, f(a))$ and $(b, f(b))$, plug it into our function, and use this to update our interval:

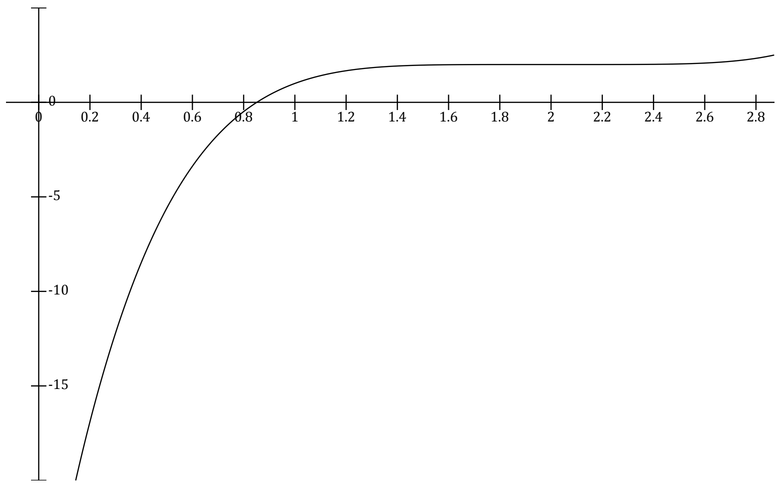
interval $[a, b]$	root $= \frac{a \cdot f(b) - b \cdot f(a)}{f(b) - f(a)}$	function at root
$[-1, 1]$	$\frac{(-1)\cdot 2 - 1 \cdot (-4)}{2 - (-4)} = \frac{1}{3}$	$f(1/3) \approx .56.$
$[-1, 1/3]$	$\frac{(-1)\cdot .56 - (1/3)\cdot(-4)}{.56 - (-4)} \approx .17$	$f(.17) \approx -.16.$

The interval $[.17, .33]$ has diameter less than twice our error tolerance. Therefore, we know that an approximation to a root of our function is the midpoint of this interval, i.e. .25.

In this example, our algorithm worked! Furthermore, it found an approximation remarkably quickly – where the bisection method would have needed 4 passes to have an accuracy of $\frac{(1-(-1))}{2^{4+1}} = \frac{1}{2^4} < \frac{1}{10}$, this needed only two passes. This is because our function is well-approximated by a line on the interval we're studying, as we can see by looking at a graph:



What happens when we try this method on something that's not particularly well-approximated by a line? For example, let's consider the function $x^5 - 10x^4 + 40x^3 - 80x^2 + 80x - 30$ on the interval $[0, 2]$.



On this interval, our function looks not particularly linear, so something awkward might happen. Nevertheless, because $f(0) = -30$ and $f(2) = 2$, we can still attempt the method of false position:

interval (a, b)	root $= \frac{a \cdot f(b) - b \cdot f(a)}{f(b) - f(a)}$	function at root
$[0, 2]$	$\frac{0 \cdot 2 - 2 \cdot (-30)}{2 - (-30)} \approx 1.86$	$f(1.86) \approx 2$
$[0, 1.86]$	$\frac{0 \cdot 2 - 1.86 \cdot (-30)}{2 - (-30)} \approx 1.74$	$f(1.74) \approx 2.$
$[0, 1.74]$	$\frac{0 \cdot 2 - 1.74 \cdot (-30)}{2 - (-30)} \approx 1.63$	$f(1.63) \approx 1.99.$
$[0, 1.63]$	$\frac{0 \cdot 1.99 - 1.63 \cdot (-30)}{1.99 - (-30)} \approx 1.53$	$f(1.53) \approx 1.98.$
$[0, 1.53]$	$\frac{0 \cdot 1.98 - 1.53 \cdot (-30)}{1.98 - (-30)} \approx 1.44$	$f(1.44) \approx 1.94.$
$[0, 1.44]$	$\frac{0 \cdot 1.94 - 1.44 \cdot (-30)}{1.94 - (-30)} \approx 1.35$	$f(1.35) \approx 1.88.$
$\vdots$	$\vdots$	$\vdots$

As you can see, this is hideously slow, and doesn't even look like it will converge at all! (In fact, you'll show this doesn't converge on the HW.) Strange.

Root-Finding Algorithms Instructor: Padraic Bartlett

Homework 1: An Introduction to Root-Finding Algorithms

Day 1 *Mathcamp 2013*

Starred problems are harder.

1. Use the bisection method to approximate a root of $x^5 - 3x^4 - 3x^2 + 2$ with an error of at most 2^{-5}.
2. Use the false-position method to approximate a root of $x^5 - x^4 - 4x + 1$ with an error of at most $\frac{1}{100}$.
3. Prove the claim we made at the end of class: that the method of false position applied to the polynomial $x^5 - 10x^4 + 40x^3 - 80x^2 + 80x - 30$ on the interval $[0, 2]$ will not converge, if we use precisely the algorithm described in class.
4. (*) Consider the following modification to the false-position method, designed to avoid the error above:

Root-Finding Algorithm 2.5: The False-Position Method, Revised

Input: A continuous function $f(x)$, along with an interval $[a, b]$ such that $f(x)$ takes on different signs on the endpoints of this interval, and an error tolerance ϵ.

(a) If $b - a < 2\epsilon$, halt. $\frac{b+a}{2}$ is an approximation to a root of $f(x)$ that is within ϵ of some actual root.

(b) Let k_a denote the number of times we've used a as an endpoint of our interval in previous runs of this algorithm if that number is greater than 0, and 1 otherwise. Define k_b similarly Set

$$c = \frac{a \cdot \frac{f(b)}{k_b} - b \cdot \frac{f(a)}{k_a}}{\frac{f(b)}{k_b} - \frac{f(a)}{k_a}}.$$

(c) Evaluate our function $f(x)$ at c. Depending on the sign of $f(c)$, choose whichever of the two intervals $[a, c]$ or $[c, b]$ our function changes sign on.

(d) Return to step 2.

Output: A value c that differs from a root of $f(x)$ by at most ϵ.

Essentially, this algorithm progressively reduces the weight of an endpoint the more times we use it, to attempt to insure that we we never wind up in the trap illustrated in question 4.

Prove that this method is guaranteed to converge. Pick one of the polynomials occuring elsewhere in this set and run this algorithm on it. Is it faster? (For reference, this algorithm is a variant of something known as the Illinois algorithm, which sees practical use in some computational settings.)