Root-Finding Algorithms — Instructor: Padraic Bartlett

# How Mathematica Does It: Newton's Method

*Day 3* — *Mathcamp 2013*

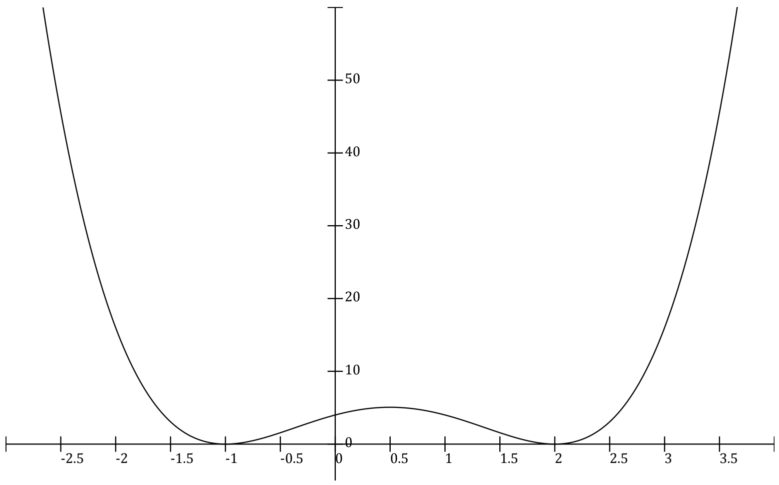
In our last two classes, we've constructed algorithms that can find all of the roots of a given functions. From a pure, non-applied, setting, this is pretty satisfying: we had a question (how do we find roots) and we created a foolproof algorithm (Sturm's theorem) that answers this completely.

However, in practice, this is only the beginning of the problem. Being **able** to find the roots, while important, is of relatively little value if our algorithm doesn't run **quickly enough** to actually be of use! In our last lecture, we're going to look at Newton's method, a root-finding technique that is how programs like Mathematica actually calculate roots.

## 1 A New Challenger Appears: Newton's Method

The motivations for Newton's method are twofold:

- In our two earlier root-finding algorithms, we needed to know the value of our function at **two** points, and furthermore know that the signs of our function differed at those two points. This is often not information we will have: i.e. for a function like $x^4 - 2x^3 - 3x^2 + 4x + 4$, there are roots (if you factor, it's $(x+1)^2(x-2)^2$) but in fact no places where the function is actually **negative**! We want to get around this issue.



- On the other hand: we saw that with the false-position method, using the data from multiple points on our curve can help us find an approximation more quickly. Effectively, we used the data at two points to create a **linear approximation** to our function, and used that to help us come up with better guesses for our roots. We want to use this observation in later algorithms!

So: let's suppose we're not in a situation where we know what's happening at two points: i.e. we have some function $f(x)$, and we're trying to find a root of $f(x)$ starting from just some initial guess $a$. How can we do this?

Well: if we use our second observation, a natural thing to try is **approximating** $f(x)$ **by a line** near this point $a$. We can do this with the derivative, by finding a line with slope $f'(a)$ through the point $(a, f(a))$:

$$f'(a) \cdot (x - a) + f(a) = y.$$

If this linear equation is a good approximation to our function, we might hope that a root of this line is close — or at least closer – to a root of $f(x)$ than our original guess $a$ was. If we solve for its root, we get

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

provided that $f'(a)$ is nonzero. This value, hopefully, is a "better guess" at a root of $f(x)$, and iterating this process should lead us to a root of $f(x)$ itself!

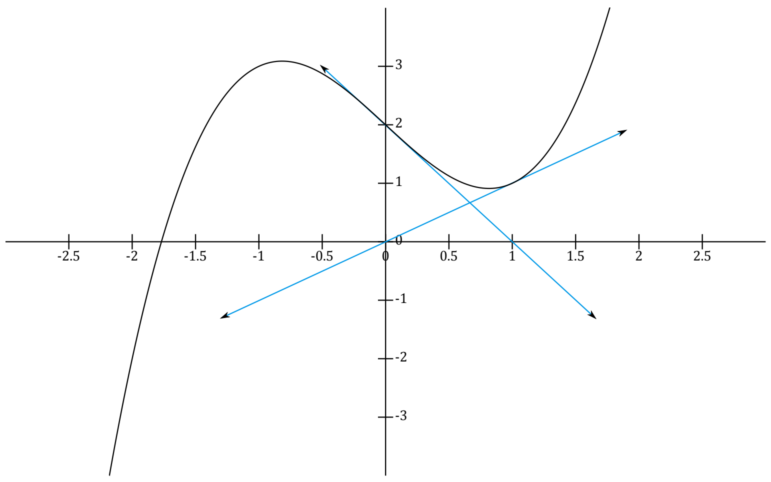
A potentially new issue with this method is that there's no particularly clear reason why this process will eventually converge on a given point. For example, consider performing this process on the function $f(x) = x^4 + x^3 - 2x^2 - 1$ starting at $x = 0$. The derivative of this function, $f'(x)$, is $4x^3 + 3x^2 - 4x$, which is 0 at $x = 0$. We can't even **start**.

Something of a problem, yes. However, this isn't such a big problem: if we just wiggle our guesses a little bit, we'll wind up with nonzero derivatives (as zeroes of derivative are rather rare if the function we're studying is a nonconstant polynomial, say.)

A larger problem, even if we can get the method started and don't run into zero derivatives, is that it still might not converge on a given value. Consider the function $x^3 - 2x + 2$, along with the starting point $x = 0$:

| starting point $a$ | $f'(a)$ | root of linear approx. |
|---|---|---|
| 0 | $-2$ | $0 - \frac{2}{-2} = 1.$ |
| 1 | 1 | $1 - \frac{1}{1} = 0.$ |
| 0 | $-2$ | $0 - \frac{2}{-2} = 1.$ |
| 1 | 1 | $1 - \frac{1}{1} = 0.$ |
| 0 | $-2$ | $0 - \frac{2}{-2} = 1.$ |
| 1 | 1 | $1 - \frac{1}{1} = 0.$ |
| $\vdots$ | $\vdots$ | $\vdots$ |

Not exactly useful. If we graph this function, we can get a visual idea for why this pathological behavior happens:



Any function with tangent lines that form the "criss-cross" pattern below, where the zeroes of various tangent lines and the initial points of other tangent lines share the same $x$-coördinate, will run into the same problem. More complicated sequences that eventually repeat can be made using this idea.

So far, so bad. However, these problems shouldn't make you give up all hope! Here's an example where the situation is somewhat rosier:

**Example.** Approximate a root to $f(x) = x^4 - 4x^3 + 2x - 1$, starting from $x = -1$.

**Answer.** First, we note that $f'(x) = 4x^3 - 12x^2 + 2$. If we apply Newton's method starting from $x = -1$, we get the following table of approximations:

| starting point $a$ | $f'(a)$ | root of linear approx. |
|---|---|---|
| $-1$ | $-14$ | $(-1) - \frac{2}{-14} = -\frac{12}{14} \approx -.85714$ |
| $-\frac{12}{14}$ | $\approx -9.33528$ | $\left(-\frac{12}{14}\right) - \frac{.34444}{-9.33528} \approx -.82025$ |
| $-.82025$ | $\approx -8.28121$ | $(-.82025) - \frac{.0196632}{-8.28121} \approx -.81788$ |
| $-.81788$ | $\approx -8.21554$ | $(-.81788) - \frac{1.14591 \cdot 10^{-4}}{-8.21554} \approx -.81787$ |
| $\vdots$ | $\vdots$ | $\vdots$ |

Further approximations are useless, as we've already — after just four runs of our algorithm! — hit as many significant digits as we're using in our calculations. By comparison, it would have taken us 17 runs of the bisection method to have found an approximation as close as this one to a root, assuming we started with an interval of length 1.

Hopefully this is motivation for why we should try to understand this method more: while it's not foolproof, it is **ridiculously** fast where it does work.

So: we really **really** want this to work. How can we insure this happens — or at least know in what situations this method is viable?

## 2 Error Analysis

A more formal way of phrasing the question above is the following: take a polynomial $f(x)$, a starting point $a_0$, and a root $a$ of $f(x)$ that we're hoping that Newton's method will converge to. Let $a_1$ denote $a_0 - \frac{f(a_0)}{f'(a_0)}$, the result of iterating Newton's method once starting from $a_0$; in general, let $a_k$ denote the result of iterating Newton's method $k$ times starting from $a_0$. We want to understand how the **error**

$$e_k := a - a_k$$

behaves as $k$ grows large. In particular, we would like to come up with conditions on $f(x)$ that insure that $e_k$ converges to 0, and furthermore get an estimate of how quickly $e_k$ converges to 0.

To start, we should attempt to describe an error $e_{k+1}$ in terms of earlier errors $e_k$, because these errors (like the approximations $a_k$ themselves) are iterative in design:

$$\begin{aligned}
e_{k+1} &= a - a_{k+1} \\
&= a - \left(a_k - \frac{f(a_k)}{f'(a_k)}\right) \\
&= (a - a_k) + \frac{f(a_k)}{f'(a_k)} \\
&= e_k + \frac{f(a_k)}{f'(a_k)}.
\end{aligned}$$

This tells us, somewhat, how to express how our errors are changing over time: we're adding $\frac{f(a_k)}{f'(a_k)}$ at each step. But this isn't necessarily very useful; we don't really know what this quantity is! We want to express $\frac{f(a_k)}{f'(a_k)}$ in terms of these errors as well.

To do this, first expand[1] our polynomial $f(x)$ around the approximation $a_k$: in other words, find constants $b_0, \ldots b_n$ such that

$$f(x) = b_0 + b_1 \cdot (x - a_k) + b_2 \cdot (x - a_k)^2 + \ldots + b_n \cdot (x - a_k)^n.$$

Because $a$ is a root of $f(x)$, we know that $f(a) = 0$. On the other hand, though, if we use our expansion, we have that

$$\begin{aligned}
f(a) &= b_0 + b_1 \cdot (a - a_k) + b_2 \cdot (a - a_k)^2 + \ldots + b_n \cdot (a - a_k)^n \\
&= b_0 + b_1 \cdot e_k + b_2 \cdot e_k^2 + \ldots b_n \cdot e_k^n.
\end{aligned}$$

What are these $b_i$'s? Well: $b_0$, in a sense, is what happens when we plug $a_k$ into our polynomial, because

$$f(a_k) = b_0 + b_1 \cdot (a_k - a_k) + b_2 \cdot (a_k - a_k)^2 + \ldots + b_n \cdot (a_k - a_k)^n = b_0.$$

[1]Typically, we write polynomials in the form $\sum_{i=1}^{n} b_i x^i$, for some constants $b_i$. However, this is not the only way to write a polynomial! For example, we can write $x^2 - 5x + 7$ as $(x-2)^2 + (x^2) + 1$. In general, given any $a$, we can write a given polynomial in the form $\sum_{i=1}^{n} c_i (x-a)^i$, by picking appropriate constants. If you haven't seen how to do this, come find me and I'll explain!

In other words, $b_0 = f(a_k)$. Similarly, $b_1$ is just what we get when we plug $a_k$ into the derivative of our function, as

$$\begin{aligned}
f'(x) &= \frac{d}{dx}\left(b_0 + b_1 \cdot (x-a_k) + b_2 \cdot (x-a_k)^2 + \ldots + b_n \cdot (x-a_k)^n\right) \\
&= b_1 \cdot \left(\frac{d}{dx}(x-a_k)\right) + b_2 \cdot \left(\frac{d}{dx}(x-a_k)^2\right) + \ldots + b_n \cdot \left(\frac{d}{dx}(x-a_k)^n\right) \\
&= b_1 \cdot \cdot (1) + b_2 \cdot (2(x-a_k)) + \ldots + b_n \cdot \left(n(x-a_k)^{n-1}\right) \\
\Rightarrow f'(a_k) &= b_1 \cdot \cdot (1) + b_2 \cdot (2(a_k-a_k)) + \ldots + b_n \cdot \left(n(a_k-a_k)^{n-1}\right) \\
&= b_1.
\end{aligned}$$

So $f'(a_k) = b_1$. Via a similar process, you can show (and should if you're skeptical!) that $f''(a_k) = 2b_2$.

Why do we care? Well: we've now created an expression relating the error terms $e_k$ to the function $f(x)$ and its derivatives at $a_k$! In particular, we've shown that

$$\begin{aligned}
f(a) &= b_0 + b_1 \cdot e_k + b_2 \cdot e_k^2 + \ldots b_n \cdot e_k^n. \\
&= f(a_k) + f'(a_k) \cdot e_k + \frac{f''(a_k)}{2} \cdot e_k^2 + O(e_k^3).
\end{aligned}$$

Here, $O(e_k^3)$ is a way of collecting together all of the terms that have three or more factors of $e_k$ in them. The idea is that if $e_k$ is something relatively small, then the terms $b_3e_k^3, b_4e_k^4, \ldots$ will all be really really small as compared to the $f(a_k)+f'(a_k)\cdot e_k+2f''(a_k)\cdot e_k^2$ factors, as they all contain additional multiples of $e_k$, a very small thing! Often in the manipulations below, we'll wind up multiplying or dividing this quantity by constants. When we do this, we won't actually write these constants down by the $O(e_k^3)$ term: i.e. we'll do things like write $2 \cdot O(e_k^3) = O(e_k^3)$. This is because

So: as we noted earlier, because $a$ is a root of $f(x)$, we have $f(a) = 0$. If we apply this to the expression above, we have

$$\begin{aligned}
0 = f(a) &= f(a_k) + f'(a_k) \cdot e_k + 2f''(a_k) \cdot e_k^2 + O(e_k^3) \\
\Rightarrow -f(a_k) &= f'(a_k) \cdot e_k + 2f''(a_k) \cdot e_k^2 + O(e_k^3) \\
\Rightarrow -\frac{f(a_k)}{f'(a_k)} &= e_k + \frac{f''(a_k)}{2f'(a_k)} e_k^2 + O(e_k^3).
\end{aligned}$$

If we plug this into our earlier expression $e_{k+1} = e_k + \frac{f(a_k)}{f'(a_k)}$ we get

$$\begin{aligned} e_{k+1} &= e_k + \frac{f(a_k)}{f'(a_k)} \\ &= e_k - \left( e_k + \frac{f''(a_k)}{2f'(a_k)} e_k^2 + O(e_k^3) \right) \\ &= -\frac{f''(a_k)}{2f'(a_k)} e_k^2 + O(e_k^3) \\ \Rightarrow |e_{k+1}| &= \left| \frac{f''(a_k)}{2f'(a_k)} e_k^2 + O(e_k^3) \right|. \end{aligned}$$

So. Suppose for the moment that on the entire region we make guesses in, we satisfy the following properties:

1. $f'(x)$ is bounded away from 0.
2. $f''(x)$ is bounded away from infinity.

As a consequence of these first two properties, we can bound $\frac{f''(a_k)}{2f'(a_k)}$ above by some constant $C$ on the interval we're studying.

We also ask for the following two other properties:

3. Our starting point $a_0$ is "close" to the root $a$. By "close," we mean that $a_0$ is sufficiently close to $a$ such that
   (a) Our $O(e_0^3)$ terms are smaller than $Ce_0^2$ term. This stops them from accidentally "taking over" our polynomial, and insures that the $\frac{2f''(a_k)}{f'(a_k)} e_k^2$ term is the relevant one to pay attention to.
   (b) $2Ce_0 < 1$. This insures that our errors don't increase when we iterate Newton's method.

If we use our third property, we can see that our initial guess $a_0$ insures that the quantity $\left| -\frac{2f''(a_0)}{f'(a_0)} e_0^2 + O(e_0^3) \right|$ is bounded above by $2C|e_0^2|$. Consequently, $e_1 \leq 2Ce_0^2 < e_0$. As a result, $e_1$ satisfies the two properties 3a, 3b as well; so by the same logic, we have $e_2 \leq 2Ce_1^2$. Induction tells us that this property holds for **all** $e_k$: i.e.

$$e_{k+1} \leq 2Ce_k^2.$$

This, by the way, is **amazing.** To get an idea of how amazing this is, think about how quickly errors improved in the method of bisections. If our error was $e_k$ at one stage, the error at the next stage was $e_k/2$, because our algorithm runs by repeatedly subdividing intervals. In a sense, this improvement is "linear" in terms of the total number of bits gained: at each step, we gain precisely one binary bit of additional information on our root.

Newton's method, by contrast, is "quadratic" in terms of the total number of bits gained. If we have an estimate that's accurate to within $10^{-5}$ of a root, running Newton just one more time gives us an improvement of $2C \cdot 10^{-5}$ on our original error margin! In other words, up to the constant multiple on the outside, we're effectively **doubling** the number of bits of accuracy on our estimate with each step. That is **ridiculous**.

And we've discovered reasonable conditions to check for to insure this happens! If we want to use Newton's method on an interval to find a root, we just need to insure that the following holds:

- $f'(x)$ is bounded away from 0 on our interval. If our root $a$ is simple, i.e. $(x-a)^2$ is not a factor of $f(x)$, this is easy to insure by simply taking a sufficiently close interval around the root we're looking for. This is because the derivative is nonzero near such a root, as we repeatedly observed in our lecture yesterday! (If our root is not simple, i.e. $(x-a)^2$ is a factor of $f(x)$, Newton's method actually is pretty slow, and we're much better off with other methods.)

- $f''(x)$ doesn't blow up to infinity, which as long as we're working with polynomials is free.

- $a_0$ is "close" to $a$, as described above: these are relatively easy conditions to check and insure by simply shrinking our interval (with i.e. the method of bisections or Sturm's theorem) until they hold.

Now that we understand when Newton's method is guaranteed to converge, we can finally formalize it below:

**Root-Finding Algorithm 4: Newton's Method**

Input: A continuous function $f(x)$ and an error tolerance $\epsilon$. As well, an interval $[c, d]$ and a guess $a_0$ that satisfy the "niceness" conditions listed above. Let $C$ be the constant defined earlier, and request that $d - c < \frac{1}{2C}$ as well.

1. Let $k$ be the current number of passes we've made using Newton's method, and $e_k$ be the error defined in our discussions above. Initially, we won't know what the root we're converging to is, but we can bound it above by $d - c$, which we know is less than $\frac{1}{2C}$. Using the same iterated argument as before, we can still conclude that our errors follow the $e_{k+1} \leq 2Ce_k^2$ property.

2. If $e_k < \epsilon$, halt: we have an approximation that's within $\epsilon$ of a root.

3. Otherwise, set $a_{k+1} = a_k - \frac{f(a_k)}{f'(a_k)}$, and return to step 2.

This, effectively, is how most modern root-finding programs actually find roots. Using this in conjunction with Sturm's theorem (to identify all the roots) and the bisection method (to improve our guesses to the point where Newton takes over) is all you need to construct fast, beautiful root-finding programs — with these three classes, you're now capable of (mostly) replicating the fastest algorithms we use today. Cool, right?

Root-Finding Algorithms Instructor: Padraic Bartlett

# Homework 3: Newton's Method

*Day 3* *Mathcamp 2013*

Starred problems are harder.

1. Use Newton's method to find a root of $x^6 - x^3 + 4x - 1$, starting at 1.
2. In class, we studied an example of Newton's method where our process kept looping between two values. (In particular, $x^3 - 2x + 2$, starting at $x = 0$, kept bouncing between $x = 0$ and $x = 1$.) Find a function $f(x)$ and a starting point $a$ such that applying Newton's method starting from $a$ enters a loop between three values.
3. Find a function $f(x)$ and a starting point $a$ so that applying Newton's method to $f(x)$ starting from $a$ yields a sequence of approximations that diverges to infinity.
4. Is it possible to find a *polynomial* $p(x)$ and starting point $a$ so that applying Newton's method to $p(x)$ starting at $a$ yields a sequence of approximations that diverges to infinity? Or must any such sequence be bounded?
5. Consider the following algorithm, the **secant method**, which is in some senses a "discrete" form of Newton's method:

Input: A continuous function $f(x)$ and a pair of guesses $a_0, a_1$ for a root of $f(x)$.

(a) To form a third guess $a_2$, draw the line between $(a_0, f(a_0))$ and $(a_1, f(a_1))$. This line has slope $\frac{f(a_1)-f(a_0)}{a_1-a_0}$, and goes through the point $(a_1, f(a_1))$. Therefore, it has the form

$$y = \frac{f(a_1) - f(a_0)}{a_1 - a_0}(x - a_0) + f(a_0).$$

(b) Solving for the root of this line tells us that our third guess, $a_2$, is precisely

$$a_2 := a_1 - f(a_1)\frac{a_1 - a_0}{f(a_1) - f(a_0)}.$$

(c) We now have a new third guess $a_2$! Take $a_1$ and $a_2$, our most recent two guesses, and repeat this process on them to generate another guess $a_3$. Iterate to generate a sequence $a_k$ of guesses.

Perform error analysis on this algorithm in a similar fashion to the one that we did for Newton's method. If $a$ is the root that the sequence $a_k$'s is hopefully converging to, and $e_k = a - a_k$, show (with some assumptions about what "niceness" means) that $|e_{k+1}| \leq C|e_k|^{\varphi}$. Note that $\varphi$, here, is the golden ratio $\frac{1+\sqrt{5}}{2}$.