Math 128a: Bisection and Newton's Method

GSI: Andrew Shi

Contents

1 Bisection

Most of us probably discovered the bisection algorithm to calculate square roots in elementary school (at least, before all kids had iPhones). Suppose I give you a 4 function calculator and ask you to approximate $\sqrt{19}$ as accurately as possible. Since $4^2 = 16$ and $5^2 = 25$, you know $\sqrt{19} \in [4, 5]$. You might consider $4.5^2 = 20.25$ next to deduce $\sqrt{19} \in [4, 4.5]$. Then you might further *bisect* this interval, notice $4.25^2 = 18.0625$, and that $\sqrt{19} \in [4.25, 4.5]$.

And then we proceed in this fashion until the interval becomes smaller and smaller after continual halving and we get an approximation to $\sqrt{19}$ to as many digits as we desire. But this is precisely the bisection algorithm for rootfinding applied to $f(x) = x^2 - 19$.

1.1 The algorithm

The bisection algorithm for rootfinding is based off the Intermediate Value Theorem, which guarantees the *existence* of a root for a continuous function f on an interval $[a, b]$ by bracketing it.

1. If $f(a), f(b)$ have opposite signs ($f(a)f(b) < 0$), then $\exists p$ such that $f(p) = 0$ for $f(x)$ a continuous function. (IVT)
2. The distance from p to the midpoint $m = \frac{a+b}{2}$ is at most $\frac{b-a}{2}$.

Let $m = \frac{a+b}{2}$ be the midpoint of $[a, b]$. In the exceedingly rare case $f(m) = 0$, then m is our root and we are done. But this pretty much never happens in floating point arithmetic for realistic problems.

Then $f(m)$ has the same sign as either $f(a)$ or $f(b)$. If $f(a)$ and $f(m)$ have the same sign, then we know the root is contained in the interval $[m, b]$, which has half the length of the original interval. Otherwise, $f(m)$ and $f(b)$ have the same sign, and we consider the interval $[a, m]$.

So we produce a sequence of nested intervals

$$p \in [a_n, b_n] \subset [a_{n-1}, b_{n-1}] \subset \cdots \subset [a_1, b_1] = [a, b]$$

with $b_n - a_n \to 0$. Practically speaking, we continue this procedure until we reach some interval with length less than some user provided tolerance ϵ, and take the midpoint of that interval to be our estimate of the root.

Example 1 *Approximate a root to $f(x) = x^2 + x - 4$ within $\epsilon = 1\text{e}{-4}$ by bisection.*

Solution: We find an interval where the function takes on opposite signs by some trial an error, say $f(0) = -4$ and $f(4) = 16$. Since $f(x)$ is a polynomial and clearly continuous on this interval, the Intermediate Value Theorem guarantees the existence of a root on this interval and we can proceed with bisection.

Table 1: Iterations of Bisection

Iter (n)	a	$f(a)$	b	$f(b)$	p_n	$f(p_n)$
1	0	-4	4	16	2	2
2	0	-4	2	2	1	-2
3	1	-2	2	2	1.5	-0.25
4	1.5	-0.25	2	2	1.75	0.8125
...	...	...	...	...	...	...
...	...	...	...	...	...	...
16	1.5615234	-0.0001211	1.5616455	0.0003821	1.5615844	0.0001305
17	1.5615234	-0.0001211	1.5615844	0.0001305	1.5615539	0.0000047

Since $|a_{16} - b_{16}| = 1.2\text{e}{-4}$, we can take the midpoint of this interval to be the approximate of the root accurate to within $\epsilon = 1\text{e}{-4}$.

Comment: The assumption that f be continuous is important to invoke the IVT. For example, consider

$$f(x) = \begin{cases} 1 & x \leq 0 \\ -1 & x > 0 \end{cases}$$

Even though we can find an interval with opposite signs, there is no guarantee of a root in the interval (and there isn't one).

1.2 Sample code and practical considerations

Here is some sample code that implements bisection.

```
function [p, k] = bisection(f, a, b, tol)

   if f(a)*f(b) > 0; disp('interval does not have opposite signs'); end

   while true
      p = (a+b)/2;
      if abs(p - a) < tol || f(p) == 0, break; end
      if f(a)*f(p) > 0
         a == p;
      else
         b == p;
      end
   end
end
```

It is useful to modify the code to terminate after a maximum number of iterations to protect against some pathologies. An obvious problem is iff the tolerance is too small (10^{-10}), which would cause the algorithm to run forever. E.g. if $a = 1$ and $b = 1 + \epsilon$ (machine ϵ), then $fl\left(\frac{a+b}{2}\right) = a$ correctly rounded, so $a \leftarrow a$ and no progress is made.

Residual based.

1.3 Convergence Analysis

TODO. Talk about how cost usually measured by number of function evaluations required because each one may be expensive. E.g. have you to do a 1M experiment to evaluate f.

2 Newton's Method

Setup: we want to solve $f(x) = 0$ and have an initial guess p_0.

2.1 Two Interpretations

Algebraic Interpretation

We want to approximate $f(x)$ by its tangent line:

$$y - f(x_0) = f'(x_0)(x - x_0)$$

Finding the root of a line is easy. Solve for x to make $y = 0$:

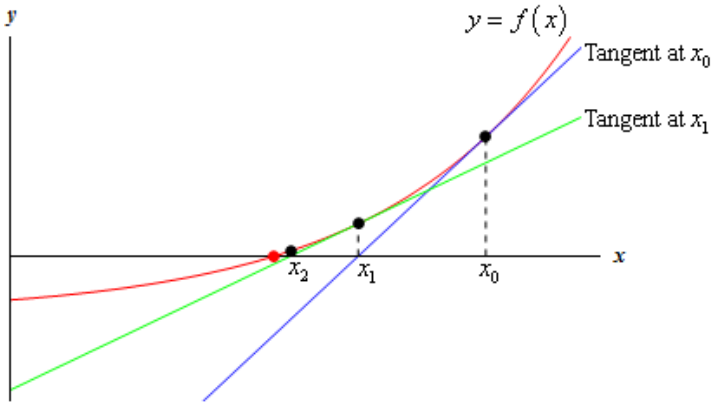
$$x = x_0 - \frac{f(x_0)}{f'(x_0)}$$

Call the result x_1. Repeat:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

In words: approximate, then solve. approximate, then solve, etc.

Graphical Interpretation



Newton's method requires an $f(x)$ that is differentiable along with an initial guess for the root x_0. Newton's method is very sensitive to this initial guess (which we will see later) and a good initial guess x_0 not easy to find, but we will usually take it as given.

Example 2 *Apply Newton's method to* $f(x) = x^2 + x - 4$ *and the initial guess* $x_0 = 2$.

Solution: We also need the derivative $f'(x) = 2x+1$. Here are the first two Newton iterations:

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = 2 - \frac{2}{5} = 1.6$$
$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)} = 1.6 - 0.038095238 = 1.56190476$$

Putting these iterations in a table:

Table 2: Iterations of Newton's Method

Iter (n)	x_n
1	1.6
2	1.56190476
3	1.56155284
4	1.56155281

Check your understanding:. How many iterations should Newton's method take to converge for a linear function $f(x)$?

2.2 Sample Code and Convergence

TODO: Can you come up with a quadratic Newton's law? If you start fast enough, Newton's method converges. Termination criteria. How to handle multiple roots.

```
f = @(x) x^2 - 2;
df = @(x) 2*x;
[p, n] = newton(f, df, 1, 1.0e-12)

function [p, n] = newton(f, df, p0, tol)
   for n = 1:100
      p = p0 - f(p0)/df(p0);
      if abs(p - p0) < tol, break; end
      p0 = p;
   end
end
```

3 Comparison of Bisection and Newton's Method

Speed

To solve $f(x) = x^2 + x - 4$, bisection needed 17 iterations and Newton's method needed only 4 iterations to converge. This is pretty typical behavior for the two algorithms. Bisection (linear convergence) is much slower than Newton's method (quadratic convergence).

Robustness

While Bisection is very slow, it is always guaranteed to find a root once the process begins. Compare this to Newton's Method, which doesn't have such guarantees. For example, consider $f(x) = x^3 - 2x + 2, x_0 = 0$. One can easily show that $x_1 = 1, x_2 = 0$, and that Newton's method will just cycle back and forth forever, even though it is clear that this function has a root. We got unlucky with this starting point – if we would have chosen $x_0 = 0.5$, we would have converged quickly to the root $x^* \approx -1.76929$.

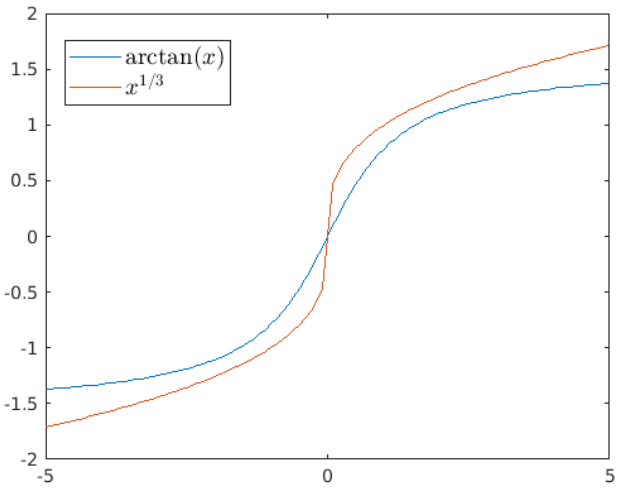
As a more pathological example, consider $f(x) = x^{1/3}$. It is clear this function has a root ($x^* = 0$). If you sketch a picture and pick any starting point, what will Newton's method do? Further away from the origin, it is clear that the slope is locally too "flat" and Newton's method will shoot you off further and further from origin with alternating signs in each iteration.

One can also show this algebraically: plugging this function into the Newton iteration gives us

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} = x_n - \frac{x^{1/3}}{\frac{1}{3}x^{-2/3}} = x_n - 3x_n = -2x_n$$

So for any nonzero initial guess, Newton's method will diverge.

Finally, let's consider $f(x) = \arctan(x)$, which is shaped like $f(x) = x^{1/3}$, exhibits rather different behavior. For $|x_0| > 1.3917$, Newton's method will diverge. For $|x_0| < 1.3917$, Newton's method will converge. For $|x_0| = 1.3917$, Newton's method will oscillate back and forth forever. This demonstrates that Newton's method is a locally convergent method - it converges if we start "close enough" to the root.



Function Evaluations vs. Derivatives

Bisection only requires function evaluations, but Newton's Method also requires a first derivative. You may take this for granted in an academic setting, but in practical situations finding the derivative can be impractical or expensive compared to a function evaluation. One way

to get around finding exact expressions for the derivative is to use a finite difference approximation for the derivative, which we will learn about in Chapter 4. It really just means take

$$f'(x_0) \approx \frac{f(x_0+h)-f(x_0)}{h} \text{ for small } h, \text{ like } h = 10^{-6}$$

Another workaround is to use the Secant Method (Poor Man's Newton Method), which approximates the derivative by the slope of a secant line. Note this requires two starting guesses which are hopefully close to the root.

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \implies x_{n+1} = x_n - \frac{f(x_{n-1})}{\frac{f(x_{n-1})-f(x_{n-2})}{x_{n-1}-x_{n-2}}}$$

Generalization to High Dimensions

This is beyond the scope of the class, but worth mentioning because it is so useful in practical situations. Oftentimes you want to solve multiple equations in multiple variables. For example:

$$\vec{\mathbf{f}}(x_1,x_2,x_3) = \begin{bmatrix} f_1(x_1,x_2,x_3) \\ f_2(x_1,x_2,x_3) \\ f_3(x_1,x_2,x_3) \end{bmatrix} = \vec{\mathbf{0}}$$

The linear approximation we use in Newton's Method has a natural generalization to higher dimensions (Math 53). The main difference is that instead of a derivative, we need a Jacobian Matrix. Here is the linearization of $\vec{\mathbf{f}}(\vec{\mathbf{x}})$ around a point $\vec{\mathbf{x_0}}$.

$$\vec{\mathbf{f}}(\vec{\mathbf{x}}) = \vec{\mathbf{f}}(\vec{\mathbf{x_0}}) + D\vec{\mathbf{f}}(\vec{\mathbf{x_0}})(\vec{\mathbf{x}} - \vec{\mathbf{x_0}})$$

Where $D\vec{\mathbf{f}} = \begin{bmatrix} \frac{\partial f_1}{x_1} & \frac{\partial f_1}{x_2} & \frac{\partial f_1}{x_3} \\ \frac{\partial f_2}{x_1} & \frac{\partial f_2}{x_2} & \frac{\partial f_2}{x_3} \\ \frac{\partial f_3}{x_1} & \frac{\partial f_3}{x_2} & \frac{\partial f_3}{x_3} \end{bmatrix} = \left[\frac{\partial f_i}{\partial x_j}\right]_{ij}$.

Bisection would require you to do a search in each direction at each iteration. Considering Bisection was slow to begin with, this additional ground to cover in the higher dimensions makes it prohibitively slow. This is an extremely important consideration in practice, and as a result Bisection is rarely used in higher dimensions.

To sum up, here is a table.

Table 3: Bisection vs. Newton's Method

Bisection	Newton
Slow (Linear)	Fast (Quadratic)
Robust	Could Fly Off
Only Function Evaluations	Needs Derivative
Only for 1D	Works well for high-D

Failures of Both Methods

And finally, here some shortcomings of both methods.

- Finding all the roots
  - Bisection: If the Intermediate Value Theorem guarantees the existence of a root in an interval $[a, b]$ and there are multiple roots in that interval, there's nothing you can really do to find all of them consistently. You will just get one of them.
  - Newton's Method: Different starting points converge to different roots (or sometimes not at all). The best you can do is try to find a good guess close to your root of interest and *hope* it converges to that.
- Repeated Roots
  - Bisection: If you have a function like $f(x) = (x-2)^2$, clearly $x = 2$ is a root, but the Intermediate Value Theorem doesn't even apply here, so you can't use Bisection to find this root.
  - Newton's Method: Newton's Method is well known to be slower (no longer quadratic convergence) in the presence of repeated roots. There are some modifications you can make to combat this, for example if you know the order of the root is n, you can try the iteration

    $$x_{n+1} = x_n - n\frac{f(x_n)}{f'(x_n)}$$

    You can read more about this in the book.