Bisection method

Monday, August 31, 2020 10:21 PM

Bisection method (aka binary search)

Main idea in pictures*

* I'm purposefully not giving pseudocode. Better to understand the idea.

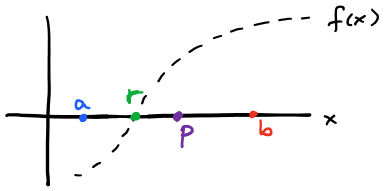
- Suppose we have a, b such that

  $\text{sign}(f(a)) = -\text{sign}(f(b))$ → "opposite signs"

  ie., $f(a) < 0$, $f(b) > 0$ or vice-versa.

  A slick way to encode this is to say $f(a) \cdot f(b) < 0$

  then by the I.V.T. (recall we assume f is continuous in this chapter), we know there is a root $r \in (a, b)$



- If we guess the midpoint $p = \frac{a+b}{2}$,

  then our error $|r - p|$ is at most half the width of the interval, $\frac{b-a}{2}$

- (*Main Idea) Recurse: $a_1 = a$, $b_1 = b$

  let's find a smaller interval $[a_2, b_2]$

  Evaluate f at the midpoint p

  If $f(a) \cdot f(p) < 0$ (ie. have opposite sign)

  $a_2 = a_1$

  $b_2 = p$ — p becomes the new right endpoint

  Else if $f(b) \cdot f(p) < 0$

  $a_2 = p$

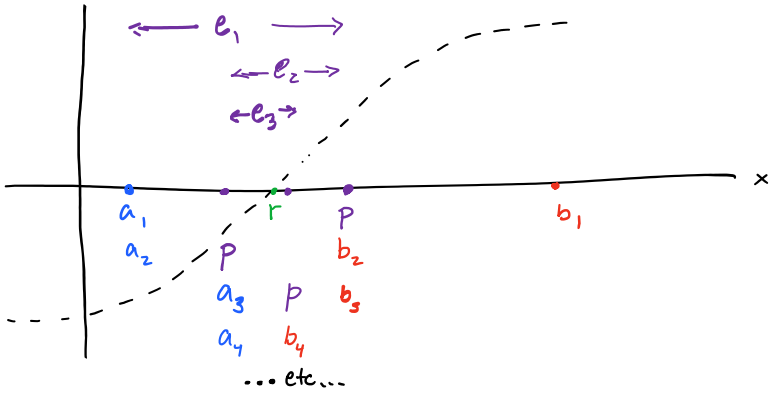
  $b_2 = b_1$ — p becomes the new left endpoint

  Else

  ... this can't happen ...

Once we recurse, and take a new midpoint, our error is at most ½ the width of our new interval: $\frac{b_2 - a_2}{2}$

... and observe this is half the error bound from the previous step.



Limitations of the method:

biggest drawbacks

- you need initial knowledge of $[a, b]$
- doesn't converge as quickly as other methods (like Newton's Method)
- doesn't easily extend to higher dimensions (whereas Newton's method does)
- requiring $f(a), f(b)$ to have opposite signs guarantees a root, but it's not necessary to have a root. So bisection doesn't always apply. Ex: $f(x) = x^2$ has a root at $x=0$ but never changes sign.

Benefits:

- doesn't need user to supply derivative f' (in fact, doesn't even require f to be differentiable)
- Converges "fast enough" usually

Practical implementation considerations

(1) Stopping conditions are tricky (even if you ignore floating point error)

If our guesses for the roots are $p_1, \dots, p_n$, then possible stopping criteria

1. $b_n - a_n < tol$ (user-supplied tolerance) (since this gives a guarantee on $|r - p|$)
2. $|p_n - p_{n-1}| < tol$ ("stagnation" of iterates)
3. $\dfrac{|p_n - p_{n-1}|}{\max(p_n, 10^{-16})} < tol$ (same as above, but relative)
4. $|f(p_n)| < tol$ (residual; doesn't give bound on error though)

5. $n < N_{max}$ (prevents ∞ loops due to bugs; this is also good if you have a time budget)

Often several criteria are used (usually with different values of tolerance) and we stop when the first criteria is met

* Note that $|p_n - p_{n-1}| \to 0$ does not imply the sequence (p_n) converges (if you know what a "Cauchy sequence" is, $|p_n - p_{n-1}| \to 0$ is not the definition of Cauchy, though it's easily confused for it)

Ex: $p_n = \sum_{k=1}^{n} \frac{1}{k}$, p_n diverges to ∞

but $p_n - p_{n-1} = \frac{1}{n} \to 0$

(2) Don't actually do $f(a) \cdot f(b) < 0$ as a check (you could get overflow!)
Instead, check the sign bit (sign in Matlab, numpy.sign in Python)
like $\text{sign}(f(a)) \cdot \text{sign}(f(b)) < 0$

(3) Mathematically, $p = \frac{a+b}{2}$ but this is unstable when $b-a$ is small
and it's not even guaranteed $p \in [a, b]$.
Better is to use the (mathematically equivalent) formula
$p = a + \frac{b-a}{2}$ which guarantees $p \geq a$

Convergence Analysis

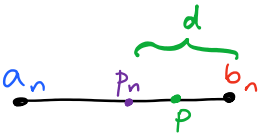
Very simple!

Thm 2.1 Let $f \in C([a_0, b_0])$ and $f(a_0) \cdot f(b_0) < 0$. Then the bisection method generates a sequence (p_n) such that
$\exists\ p \in (a_0, b_0)$ with $f(p) = 0$ and $p_n \to p$ and $|p_n - p| \leq \frac{b_0 - a_0}{2^n}$ $(\forall n \geq 1)$

proof Recall $p_n = \frac{1}{2}(a_n + b_n)$ and we argued that $b_n - a_n = \frac{1}{2}(b_{n-1} - a_{n-1})$, so

$$b_n - a_n = \frac{1}{2^{n-1}}(b_0 - a_0)$$

and $|p_n - p| \le \frac{1}{2}(b_n - a_n)$ (d)



So putting it all together, $|p_n - p| \le \frac{1}{2^n}(b_0 - a_0)$

which implies $p_n \to p$. $\square$

What kind of convergence is this?

If error $e_n := |p_n - p|$ decays like $e_n \le \text{const} \cdot \rho^n$ for $\rho < 1$,

this is linear convergence. And indeed bisection has linear convergence for const $= (b_0 - a_0)$ and $\rho = \frac{1}{2}$.

Not as fast as Newton*, but pretty fast

*Convergence for Newton's Method will only be fast when we get close enough to the solution. For low/moderate accuracy, bisection can be better.