Bisection vs Anderson–Björck:

A Comparison of Root-finding Methods

Matt D'Urso

Aug 3, 2026

Colgate University

Intro

These slides cover 4 root-finding methods:

1. Bisection
2. Regula falsi “method of false position”
3. Illinois method (Snyder, 1953; Dowell & Jarratt, 1971)
4. Anderson–Björck (Anderson & Björck, 1973)

I will assume you know the concept of bisection, so that will be a quick review/reference point for us

Motivation

Why even do this?-

Well, why even do anything? If you're going to do something, do it well.

More importantly, when it's 3:30am and you're trying to speed through a calibration, each iteration takes years off your life and decades off your rapidly dwindling sanity

- You'd like to shrink the initial brackets, but when targeting strong asymmetries (e.g. size distributions), things don't behave as nicely as you'd like
- Widening the brackets adds 1, maybe 2 iters, but you are no longer a sane person that properly allocates your time

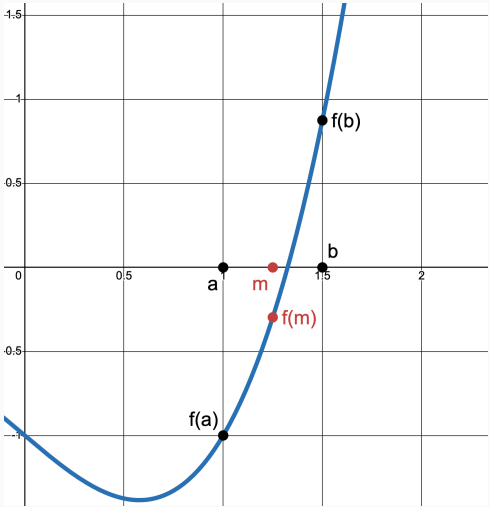
Bisection Review

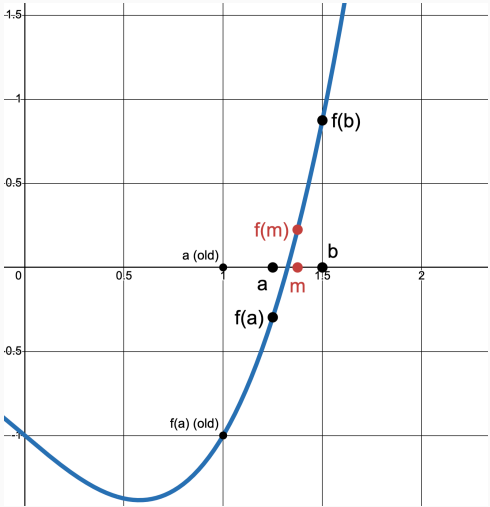
Suppose I wanted to find the root of $f(x)$, which is continuous between a and b:

1. Pick initial brackets (a and b) where $f(a)$ and $f(b)$ have opposite signs
2. Find the midpoint of a and b, $m = \frac{a+b}{2}$ and take $f(m)$
3. If $f(m)$ has the same sign as $f(a)$, replace a with m
   - Or, replace b with m if $f(m)$ has the same sign as $f(b)$
4. Continue until $|a - b| < desired\_tol$. Then m sufficiently approximates the root

Next slide: example with $f(x) = x^3 - x - 1$ and initial $a = 1$ and $b = 1.5$

Bisection: Steps 1-3 (All graphs made with Desmos)





Bisection: Steps 4 - N

Now $m = 1.375$ becomes the new b and the process repeats until convergence

- Determined by $|a - b| < \epsilon$, where ϵ is sufficiently small and chosen by the modeler

This is a perfectly valid method, but we can make it faster with the “false position”

Regula Falsi

We use bisection, or various other numerical methods, because economics is hard

- What is simple is finding the root of a line

We can use the root of a line to better update our new a or b point:

1. Instead of the midpoint of a and b, draw a line between $f(a)$ and $f(b)$
2. Call the root of this line c and take $f(c)$
   - I'm using c since this is no longer a midpoint, but call it m if you like
3. If $f(c)$ has the same sign as $f(a)$, replace a with c
   - Or, replace b with c if $f(c)$ has the same sign as $f(b)$
4. Continue until $|a - b| < desired\_tol$. Then c approximates the root (more latter?)

Building the Line

To draw our line, linearly interpolate (and rearrange a bit) $f(a)$ and $f(b)$:

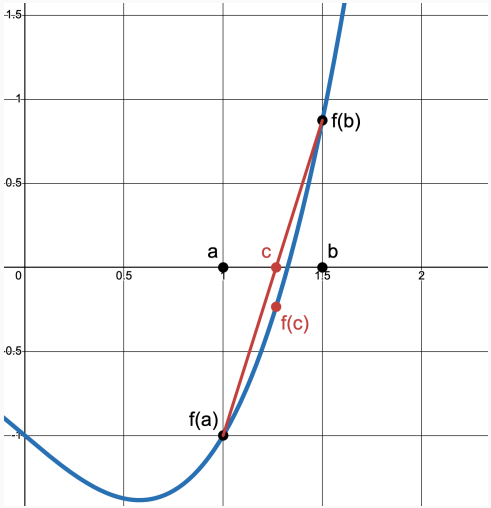
$$c = b - \frac{f(b) \times (b - a)}{f(b) - f(a)}$$

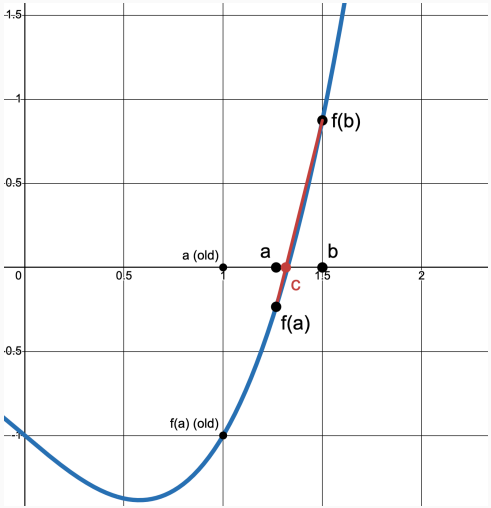
Optional: set a condition to use the smaller $|f(val)|$. If $|f(a)| < |f(b)|$, use:

$$c = a - \frac{f(a) \times (b - a)}{f(b) - f(a)}$$

- This likely won't matter outside of floating-point dust

Regula Falsi: Steps 1-4

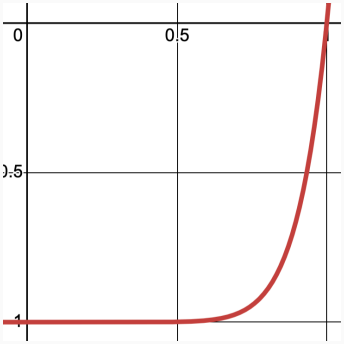




Issues?

Consider the case of $f(x) = x^{10} - 1$ on $[0, 1.3]$

- $f(x)$ is flat near 0 and blows up near 1.3
- Our line always crosses the axis around the left end
- $b = 1.3$ is retained forever and a crawls towards 1 (true root)
- *Why can't $|a - b|$ work here? Hint: think about* 0.3



The Illinois Method

The Illinois method aims to correct this issue

- First as technical note by Snyder (1953), then formalized by Dowell & Jarratt (1971)

WLOG, if sign $f(c)$ = sign $f(a)$, then the root is in $[c, b]$

- c is the new a; b remains the same
- No difference from regula falsi yet

The change comes from how we "dislodge" $f(b)$...

The Illinois Insight

We need $f(b)$ to move closer to 0 so we just halve it

Take the same regula falsi algorithm we already built, modified with:

- If sign $f(c)$ = sign $f(a)$ twice in a row, $f(b) = \frac{f(b)}{2}$
- If it happens again on the next iter, halve again...
- *Everything else remains the same as in regula falsi*
- Convergence guaranteed (halving does not change sign; stale bracket replaced)
  - Order of convergence ≈ 1.442 (Dowell & Jarratt, 1971)

Note: some always halve the non-moving bound on every iteration. I haven't seen a noticeable difference when I do this.

Regula Falsi Failure and Illinois Correction for $f(x) = x^{10} - 1$

Regula Falsi:

```
iter=1  a=0.0  b=1.3  c=0.0942995953723274
iter=2  a=0.0942995953723274  b=1.3  c=0.18175887251907943
iter=3  a=0.18175887251907943  b=1.3  c=0.26287401252030435
iter=4  a=0.26287401252030435  b=1.3  c=0.3381051033222695
iter=5  a=0.3381051033222695  b=1.3  c=0.4078779165927524
iter=6  a=0.4078779165927524  b=1.3  c=0.47258315356239
iter=7  a=0.47258315356239  b=1.3  c=0.5325715106320142
iter=8  a=0.5325715106320142  b=1.3  c=0.5881445691706799
iter=9  a=0.5881445691706799  b=1.3  c=0.6395439707682159
iter=10  a=0.6395439707682159  b=1.3  c=0.6869431667412391
iter=11  a=0.6869431667412391  b=1.3  c=0.7304464366929381
iter=12  a=0.7304464366929381  b=1.3  c=0.7700987444968075
iter=13  a=0.7700987444968075  b=1.3  c=0.8059075090169241
iter=14  a=0.8059075090169241  b=1.3  c=0.8378738914244527
iter=15  a=0.8378738914244527  b=1.3  c=0.8660276320873561
iter=16  a=0.8660276320873561  b=1.3  c=0.8904572813509999
iter=17  a=0.8904572813509999  b=1.3  c=0.911328251308228
iter=18  a=0.911328251308228  b=1.3  c=0.9288846695992143
iter=19  a=0.9288846695992143  b=1.3  c=0.9434360517492835
iter=20  a=0.9434360517492835  b=1.3  c=0.9553339751921469
iter=21  a=0.9553339751921469  b=1.3  c=0.9649455723416893
iter=22  a=0.9649455723416893  b=1.3  c=0.9726296888054463
iter=23  a=0.9726296888054463  b=1.3  c=0.9787191336334624
```

*Max iter hit at 100 using $|f(c)| < \epsilon = 1e\text{-}12$

Illinois:

```
iter=1  a=0.0  b=1.3  c=0.0942995953723274
iter=2  a=0.0942995953723274  b=1.3  c=0.18175887251907943
iter=3  a=0.18175887251907943  b=1.3  c=0.3330171567671213
iter=4  a=0.3330171567671213  b=1.3  c=0.5634423147022629
iter=5  a=0.5634423147022629  b=1.3  c=0.8463635731395354
iter=6  a=0.8463635731395354  b=1.3  c=1.074910177068493
iter=7  a=0.8463635731395354  b=1.074910177068493  c=0.9454923183277778
iter=8  a=0.9454923183277778  b=1.074910177068493  c=0.9828011093189349
iter=9  a=0.9828011093189349  b=1.074910177068493  c=1.0040954923602121
iter=10  a=0.9828011093189349  b=1.0040954923602121  c=0.9996755374546944
iter=11  a=0.9996755374546944  b=1.0040954923602121  c=0.9999940615677148
iter=12  a=0.9999940615677148  b=1.0040954923602121  c=1.000005704633256
iter=13  a=0.9999940615677148  b=1.000005704633256  c=0.9999999998475554
iter=14  a=0.9999999998475554  b=1.000005704633256  c=0.9999999999999961
```

The Anderson–Björck Adjustment

The Illinois method is great, but it can be sped up with a dynamic adjustment factor

WLOG, if sign $f(c) = \text{sign } f(a)$, Anderson and Björck use $k \times f(b)$, where:

$$k' = 1 - \frac{f(c)}{f(a)},$$

$$k = \begin{cases} k' & \text{if } k' > 0, \\ \frac{1}{2} & \text{otherwise.} \end{cases}$$

Updating here happens each time, not after two matching signs in a row

- *Everything else still remains the same as in regula falsi*
  - Order of convergence ≈ 1.7 (Anderson & Björck, 1973)

Analyzing the Steps

From before:

$$k' = 1 - \frac{f(c)}{f(a)}, \qquad k = \begin{cases} k' & \text{if } k' > 0, \\ \frac{1}{2} & \text{otherwise.} \end{cases}$$

- Dynamism here allows us to adjust based on the relative distance of y vals

If $k' < 0$, we want to avoid using it and "artificially" changing a sign with this weight

- *But what does this tell us about the step we just took?*
- Fallback is simply Illinois

My JMP D-StSt: Bisection vs Anderson–Björck

Bisection:

```
================================================================
  METHOD 1: BISECTION    bracket [5, 10],  xtol = 1e-07 on the price
================================================================
iter          pval          wage             C       g=C-1/p
   1     5.0000000      0.428000      0.030432     -1.70e-01
   2    10.0000000      0.214000      0.568592     +4.69e-01
   3     7.5000000      0.285333      0.167265     +3.39e-02
   4     6.2500000      0.342400      0.077432     -8.26e-02
   5     6.8750000      0.311273      0.115721     -2.97e-02
   6     7.1875000      0.297739      0.139706     +5.76e-04
   7     7.0312500      0.304356      0.127275     -1.49e-02
   8     7.1093750      0.301011      0.133378     -7.28e-03
   9     7.1484375      0.299366      0.136514     -3.38e-03
  10     7.1679688      0.298550      0.138103     -1.41e-03
  11     7.1777344      0.298144      0.138903     -4.17e-04
  12     7.1826172      0.297942      0.139304     +7.92e-05
  13     7.1801758      0.298043      0.139104     -1.69e-04
  14     7.1813965      0.297992      0.139204     -4.48e-05
  15     7.1820068      0.297967      0.139254     +1.72e-05
  16     7.1817017      0.297980      0.139229     -1.38e-05
  17     7.1818542      0.297973      0.139241     +1.69e-06
  18     7.1817780      0.297976      0.139235     -6.07e-06
  19     7.1818161      0.297975      0.139238     -2.19e-06
  20     7.1818352      0.297974      0.139240     -2.52e-07
  21     7.1818447      0.297974      0.139241     +7.17e-07
  22     7.1818399      0.297974      0.139240     +2.32e-07
  23     7.1818376      0.297974      0.139240     -1.02e-08
  24     7.1818388      0.297974      0.139240     +1.11e-07
  25     7.1818382      0.297974      0.139240     +5.04e-08
  26     7.1818379      0.297974      0.139240     +2.01e-08
  27     7.1818377      0.297974      0.139240     +4.95e-09
  28     7.1818376      0.297974      0.139240     -2.62e-09

  METHOD 1: BISECTION:  p = 7.1818376    28 solves total    161.22 seconds
```

Anderson–Björck:

```
================================================================
  METHOD 3: ANDERSON-BJORK    bracket [5, 10],  xtol = 1e-07 on the price
================================================================
iter          pval          wage             C       g=C-1/p
   1     5.0000000      0.428000      0.030432     -1.70e-01
   2    10.0000000      0.214000      0.568592     +4.69e-01
   3     6.3285720      0.338149      0.081606     -7.64e-02
   4     7.1688375      0.298514      0.138174     -1.32e-03
   5     7.1835141      0.297904      0.139378     +1.70e-04
   6     7.1818333      0.297974      0.139240     -4.41e-07
   7     7.1818377      0.297974      0.139240     +1.85e-11
   8     7.1818377      0.297974      0.139240     -1.67e-16

  METHOD 3: ANDERSON-BJORK:  p = 7.1818377    8 solves total    46.01 seconds
```

*Illinois took roughly 1 minute.

Other Methods?

Many other weights exist, along with a multitude of completely different methods, which may be faster

HOWEVER, Anderson–Björck has three benefits in one: speed, [relative] ease of learning, and guaranteed to work

On Brent (1971), I think this is what ‘fzero’ does in MATLAB, but YOU’RE not using pre-made computations anyway, right?? Regardless, the decision tree you’ll have to follow for Brent is annoying and the path can move a lot with different warm starts. Not something your K-S regression coefficients will like without a strong tolerance. You do not want to debug this at 4:00am.

References

- Anderson, N. & Å. Björck (1973). **“A New High Order Method of Regula Falsi Type for Computing a Root of an Equation”.** In: *BIT* 13.3, pp. 253–264. DOI: *10.1007/BF01951936*.
- Brent, R. P. (1971). **“An Algorithm with Guaranteed Convergence for Finding a Zero of a Function”.** In: *The Computer Journal* 14.4, pp. 422–425. DOI: *10.1093/comjnl/14.4.422*.
- Dowell, Mark & Peter Jarratt (1971). **“A Modified Regula Falsi Method for Computing the Root of an Equation”.** In: *BIT* 11.2, pp. 168–174. DOI: *10.1007/BF01934364*.
- Snyder, James N. (1953). ***Inverse interpolation, a real root of*** $f(x) = 0$**.** ILLIAC I Library Routine H1-71. University of Illinois Digital Computer Laboratory, p. 4.

Graphs: https://www.desmos.com/calculator