

✓ 01.05 rootfinding: without derivatives

what if $f(x)$ has no (or unknown) $f'(x)$?

✓ 1 secant method, variants

✓ secant method

replace the derivative with a difference quotient. ie, replace tangent line with secant line through previous two guesses. ie, approximation for derivative at x_i is difference quotient

$$\frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}.$$

secant method

$x_0, x_1 = \text{initial guesses}$

$$x_{i+1} = x_i - \underbrace{f(x_i) \cdot \frac{x_i - x_{i-1}}{f(x_i) - f(x_{i-1})}}_{\sim \frac{1}{f'(x_i)}}, \quad i = 1, 2, 3, \dots$$

✓ algorithm

```
icount = 0

fx_old = f(x_old)
if abs(fx_old) < epsilon # epsilon ~ eta
    return x_old
fx_older = f(x_older)
if abs(fx_older) < epsilon # epsilon ~ eta
    return x_older

dq = (fx_old - fx_older)/(x_old - x_older)
x_new = x_old - fx_old/dq
fx = f(x_new)
icount = icount + 1
```

```
# while (abs(fx) > epsilon) and (icount <= imax): # epsilon ~ eta
while (abs(x_new - x_old) > epsilon) and (icount <= imax):
    x_older = x_old
    fx_older = fx_old
    x_old = x_new
    fx_old = fx
    dq = (fx_old - fx_older)/(x_old - x_older)
    x_new = x_old - fx_old/dq
    fx = f(x_new)
    icount = icount + 1

return x_new
```

✓ convergence

assume that method converges to r and $f'(r) \neq 0$, then the approximate error relationship

$$e_{i+1} \approx \left| \frac{f''(r)}{2f'(r)} \right| e_i e_{i-1}$$

holds and implies

$$e_{i+1} \approx \left| \frac{f''(r)}{2f'(r)} \right|^{\alpha-1} e_i^\alpha,$$

where $\alpha = \frac{1+\sqrt{5}}{2} \approx 1.62$. secant method convergence to simple roots is called **superlinear**, meaning that it lies between linearly and quadratically convergent methods.

✓ example 16

example 01, revisted. apply secant method with $x_0 = 0, x_1 = 1$ to find root of $f(x) = x^3 + x - 1$.

$$x_{i+1} = x_i = \frac{(x_i^3 + x_i - 1)(x_i - x_{i-1})}{x_i^3 + x_i - (x_{i-1}^3 + x_{i-1})}$$

$$\Downarrow \quad x_0 = 0, x_1 = 1$$

$$x_2 = 1 - \frac{(1)(1 - 0)}{(1 + 1 - 0)} = \frac{1}{2}$$

$$x_3 = \frac{1}{2} - \frac{-\frac{3}{8}(\frac{1}{2} - 1)}{\frac{3}{8} - 1} = \frac{7}{11}.$$

▼ code, example 16

```

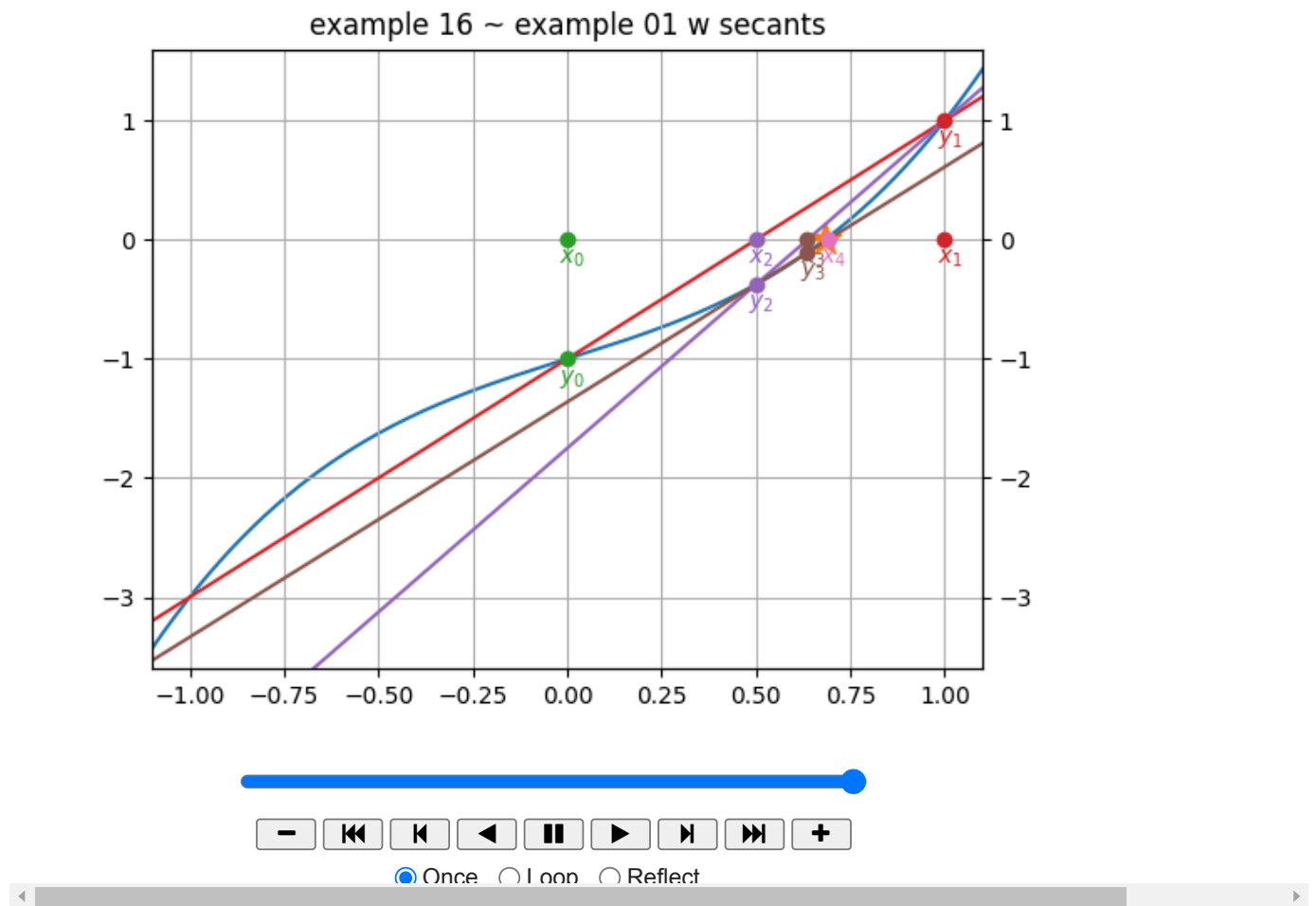
1 # example 16, secant method for example 01 # mod example 14
2
3 if __name__ == "__main__":
4

```

⇌ expected convergence: [0.8068743]

example 16 ~ example 01 w secants. x0 = [0, 1].

i	x[i]	e[i]	"/e[i-1]^α
000	0.000000000000000	0.68232780382802	
001	1.000000000000000	0.31767219617198	0.58963609730649
002	0.500000000000000	0.18232780382802	1.16591656728344
003	0.636363636363636	0.04596416746438	0.72174623555971
004	0.69005235602094	0.00772455219292	1.12751982884733
005	0.68202041964819	0.00030738417983	0.80384864065885
006	0.68232578140989	0.00000202241813	0.97481374069426
007	0.68232780435903	0.000000000053101	0.86774886684321
008	0.68232780382802	0.000000000000000	1.01567939794005
009	0.68232780382802	0.000000000000000	207243987.96379616856575



✓ generalizations of secant method

✓ **regula falsi**

aka "method of false position". regula falsi is similar to bisection but midpoint replaced by secant-like approximation. ie, given bracketing interval $[a, b]$,

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

where $c \in [a, b]$ and next subinterval chosen to bracket root.

✓ algorithm, regula falsi

given $[a, b]$ st $f(a) \cdot f(b) < 0$

for $i = 1, 2, 3, \dots$

```

c = [b·f(a) - a·f(b)] / [f(a) - f(b)]
if f(c) == 0 stop
if f(a)·f(c) < 0
    b = c
else
    a = c
end
next

```

✓ code, regula falsi

```

1 # algorithm, basic
2
3 def secant_rf(f,ab,tol=1e-8):
4

1 # algorithm, basic # modified bisection_expanded from lecture 01_01
2
3 def secant_rf_expanded(f,ab,tol=1e-8,all=False,workspace=False):
4

```

✓ example 17

apply regula falsi on interval $[-1, 1]$ to find root $r = 0$ of $f(x) = x^3 - 2x^2 + \frac{3}{2}x$.

$$x_0 = -1, x_1 = 1$$

$$x_2 = \frac{x_1 \cdot f(x_0) - x_0 \cdot f(x_1)}{f(x_0) - f(x_1)} = \frac{1(-\frac{9}{2}) - (-1)(\frac{1}{2})}{-\frac{9}{2} - \frac{1}{2}} = \frac{4}{5}.$$

$f(-1) \cdot f(\frac{4}{5}) < 0 \implies [x_0, x_2] = [-1, \frac{4}{5}] \sim$ better than $\frac{1}{2}$ of bisection; however, sometimes its not your birthday.

✓ code, example 17, regula falsi

```

1 # requires prior execution of secant_rf_expanded() # mod lecture 01_01 first code
2
3 if __name__ == "__main__":
4

```

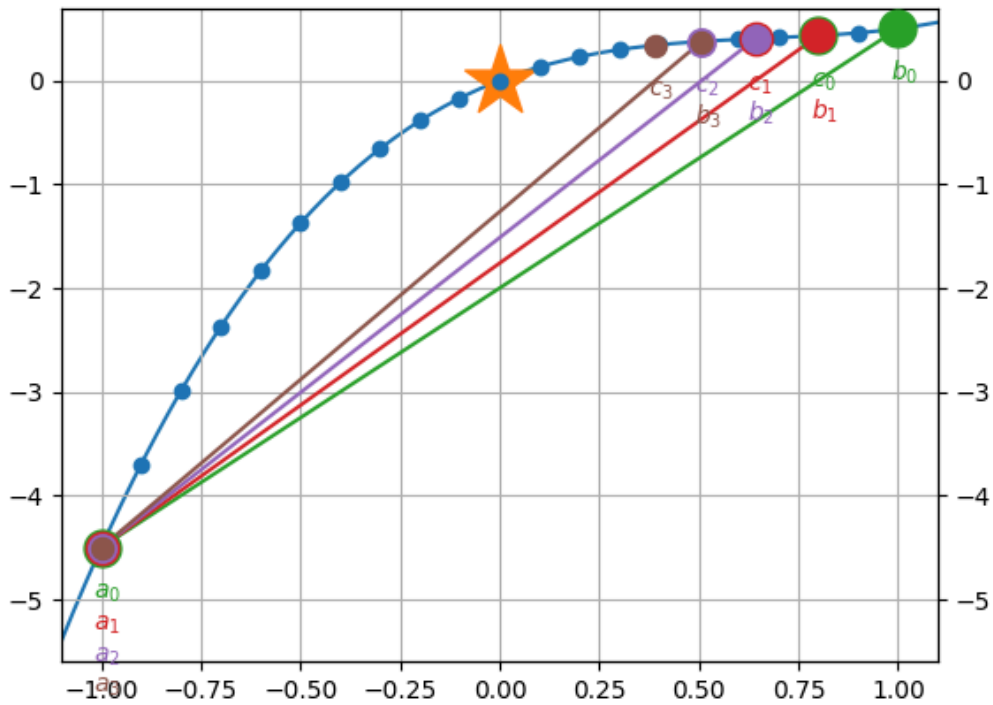
➡ regular falsi: $x^3 + 2x^2 - 1.5x$, $x \in [-1.0, 1.0]$

i	a	f(a)	b	f(b)	c	f(c)	±
000	-1.00000000	-4.50000000	1.00000000	0.50000000	0.80000000	0.43200000	001
001	-1.00000000	-4.50000000	0.80000000	0.43200000	0.64233577	0.40333785	001
002	-1.00000000	-4.50000000	0.64233577	0.40333785	0.50724082	0.37678437	001
003	-1.00000000	-4.50000000	0.50724082	0.37678437	0.39079015	0.34043162	001

1 ani



regular falsi: $x^3 + 2x^2 - 1.5x$, $x \in [-1.0, 1.0]$



☒ Once ☐ Loop ☐ Reflect

✓ code, example 17, secant

```
1 # example 17, secant method # mod example 16
2
3 if __name__ == "__main__":
164
```



expected convergence: [1.19458315]

secant: $x^3 + 2x^2 - 1.5x$, $x \in [-1.0, 1.0]$

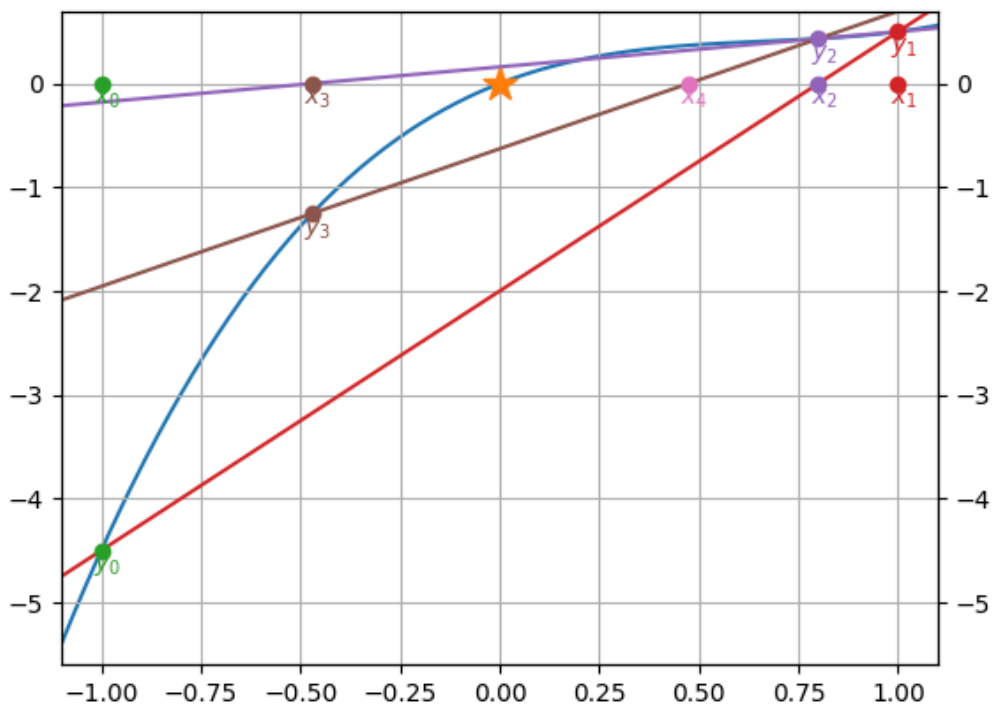
i	x[i]	e[i]	"e[i-1] ^α
000	-1.0000000000000000	1.0000000000000000	
001	1.0000000000000000	1.0000000000000000	1.0000000000000000
002	0.8000000000000000	0.8000000000000000	0.8000000000000000
003	-0.47058823529412	0.47058823529412	0.67521916496024
004	0.47424724729948	0.47424724729948	1.60576763820552
005	0.25965464146786	0.25965464146786	0.86822303833010
008	0.03994345289533	0.03994345289533	1.49983922882677
009	-0.00643218295752	0.00643218295752	1.17833000297553
010	0.00035223951983	0.00035223951983	1.23876434915084
011	0.00000300563144	0.00000300563144	1.16216844505921

| 012 | -0.00000000141202 | 0.00000000141202 | 1.21542677765262 |

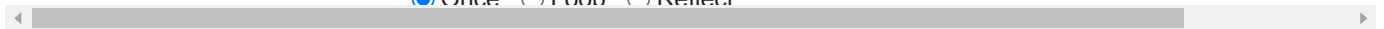
1 ani



secant: $x^3 + 2x^2 - 1.5x$, $x \in [-1.0, 1.0]$



☒ Once ☐ Loop ☐ Reflect



✓ mullers method

draw parabola $y = p(x)$ through three previous points (vs line through two previous points) and its intersection with x -axis closest to x_i is next iteration x_{i+1} .

- for multiple intersections, select the one closest to previous iteration;
- if parabola misses x -axis then it gets complex and costs extra tuition. 🙄

oscar velize [@youtube](#)

✓ code, mullers method

```
1 # https://en.wikipedia.org/wiki/Muller%27s_method # mod
2
3 import cmath as cm
4 import numpy as np
5
```

```

6 # newtons divided difference
7 def dd(f,xx):
8     if len(xx) == 2:
9         a,b = xx

1 # instead of complex number hack, use future methods
2
3 import numpy as np
4
5 def mullers_lol(f,xs,max_iter=100,tol=1e-8,method=0):
6     """

1 # example 01 with mullers # mod secant example
2
3 if __name__ == "__main__":...
153
154

```

➡ step 0, parabola: $-1.0 + 3.0 \cdot x - 1.0 \cdot x^2$
 roots: [2.6180339887498976, 0.38196601125010493]

step 1, parabola: $-1.0 - 6.23606798 \cdot x + 3.61803399 \cdot x^2$
 roots: [1.871307386268059, -0.14770058851807896]

step 2, parabola: $8.79829268 - 14.87782909 \cdot x + 5.48934138 \cdot x^2$
 roots: [1.8385321777112726, 0.8717800572671759]

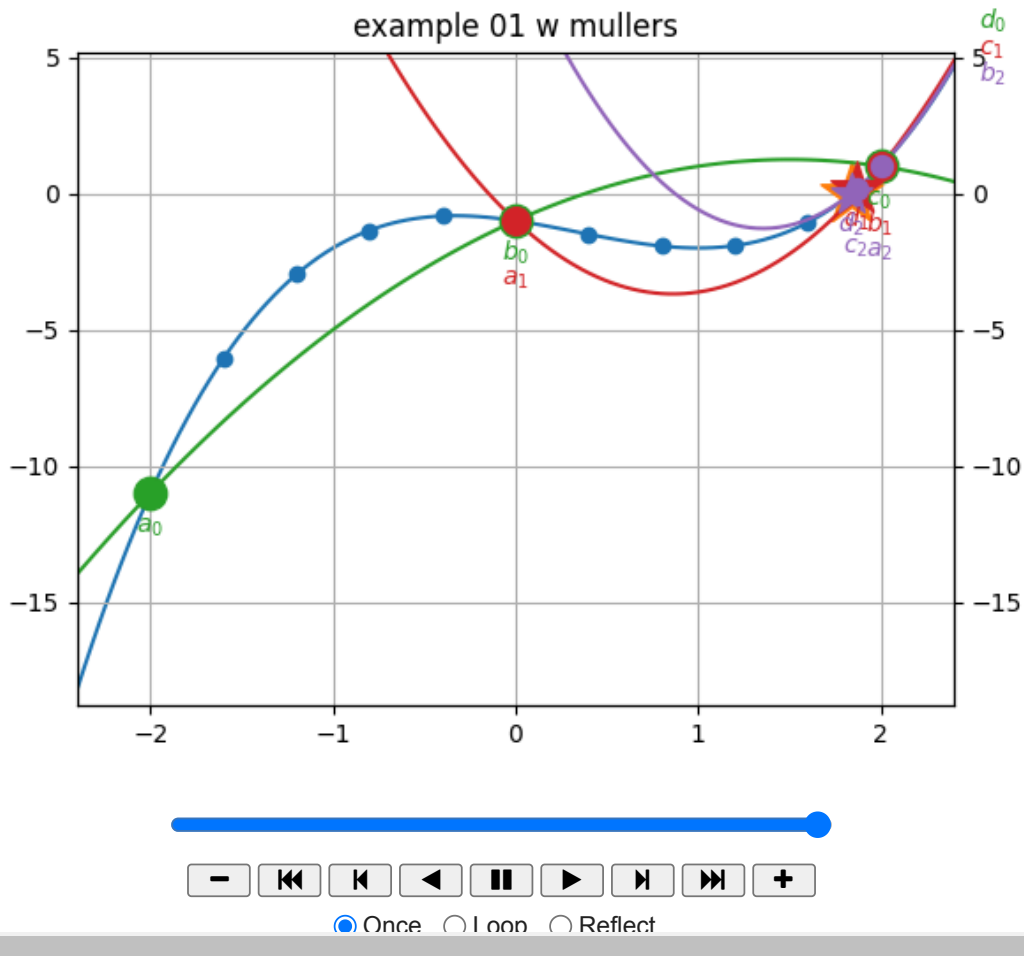
step 3, parabola: $8.00723819 - 14.15294492 \cdot x + 5.32787355 \cdot x^2$
 roots: [1.8392902102200412, 0.8171063380988388]

step 4, parabola: $5.32800227 - 11.26393044 \cdot x + 4.54912977 \cdot x^2$
 roots: [1.8392867552294225, 0.6367759193327744]

example 01 w mullers: $x^3 - x^2 - x - 1$

i	a	b	c	gap	e[i]	"/e[i-1]
000	-2.00000000	0.00000000	2.00000000	2.00000000	0.16071324	
001	0.00000000	2.00000000	2.61803399	0.61803399	0.77874723	4.84556973
002	2.00000000	2.61803399	1.87130739	-0.74672660	0.03202063	0.04111813
003	2.61803399	1.87130739	1.83853218	-0.03277521	0.00075458	0.02356535
004	1.87130739	1.83853218	1.83929021	0.00075803	0.00000346	0.00457873
005	1.83853218	1.83929021	1.83928676	-0.00000345	0.00000000	0.00000442

1 ani



- ✓ **inverse quadratic interpolation (IQI)**

similar to mullers but with parabola $x = p(y)$, which is handy for limiting the x -axis intersection to a single point.

consider second-degree polynomial $x = P(y)$ through points $(a, A), (b, B), (c, C)$.

$$P(y) = a \frac{(y-B)(y-C)}{(A-B)(A-C)} + b \frac{(y-A)(y-C)}{(B-A)(B-C)} + c \frac{(y-A)(y-B)}{(C-A)(C-B)}$$

$$\Downarrow \quad P(A) = a, P(B) = b, P(C) = c, y = 0$$

$$P(0) = c - \frac{r(r-q)(c-b) + (1-r)s(c-a)}{(q-1)(r-1)(s-1)}, \quad q = \frac{f(a)}{f(b)}, r = \frac{f(c)}{f(b)}, s = \frac{f(c)}{f(a)}$$

$$\Downarrow \quad a = x_i, b = x_{i+1}, c = x_{i+2}, A = f(x_i), B = f(x_{i+1}), C = f(x_{i+2})$$

$$x_{i+3} = x_{i+2} - \frac{r(r-q)(x_{i+2} - x_{i+1}) + (1-r)s(x_{i+2} - x_i)}{(q-1)(r-1)(s-1)}, \quad q = \frac{f(x_i)}{f(x_{i+1})}, r = \frac{f(x_{i+2})}{f(x_{i+1})}, s = \frac{f(x_{i+2})}{f(x_i)}.$$

here x_{i+3} replaces x_i but an alternative implementation replaces the largest source of backward error.

lemonfully [@youtube](#) [@wiki](#)

- lemonfully points out that while this method is asymptotically faster than secants, it only is if initial points chosen well.
- oscar veliz (in the lead up to [brents method](#)) also points out this unreliability.

✓ oh, why not

```
1 # a few reasons come to mind
2
3 import numpy as np
4 import statistics as st
5
6 def iqi(f,xs,max_iter=100,tol=1e-8,doyourworst=False,workspace=False):
7
```

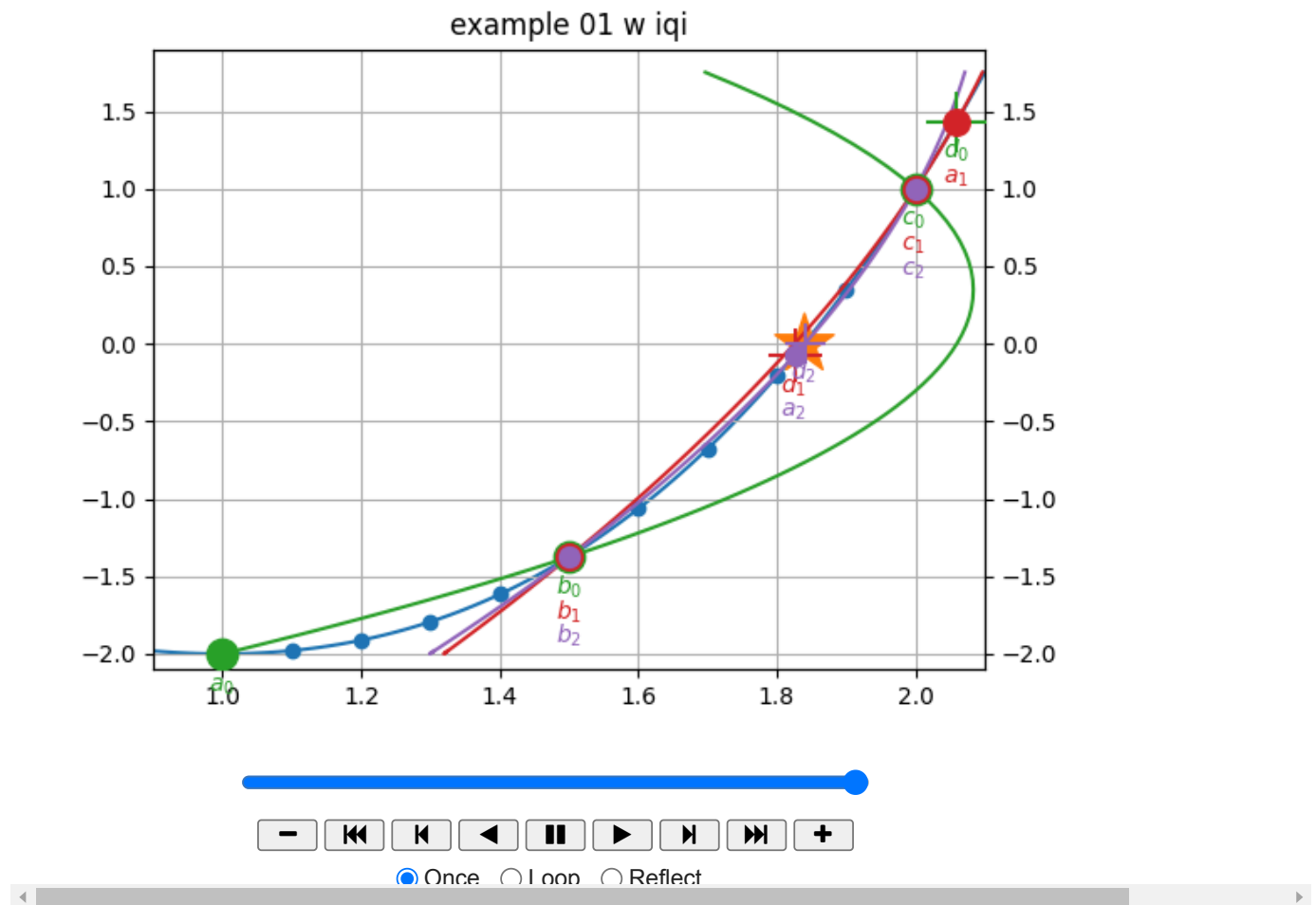
```
1 if __name__ == "__main__": ...
166
```

➡ root: [1.83928676]

example 01 w iqi: $x^3 - x^2 - x - 1$

i	a	b	c	bwe (Δy)	e[i]	"/e[i-1]
000	1.00000000	1.50000000	2.00000000	0.50000000	0.83928676	
001	2.05964912	1.50000000	2.00000000	0.43554618	0.22036237	0.26255909
002	1.82546813	1.50000000	2.00000000	1.07473271	0.01381863	0.06270865
003	1.82546813	1.84037070	2.00000000	0.08066758	0.00108394	0.07844079
004	1.82546813	1.84037070	1.83928426	0.00594852	0.00000250	0.00230327
005	1.83928676	1.84037070	1.83928426	0.00001366	0.00000000	0.00001790
006	1.83928676	1.83928676	1.83928426	0.00000000	0.00000000	0.00000497

```
1 ani
2
```



```
1 # briefly
2
3 if __name__ == "__main__":
4
5     f = lambda x: pow(x,3) + x - 1
6
7     if True:
8         x = iqi(f,[0.,0.5,1.])
9         print(f"iqi, std: {x}\n")
10
11     x = iqi(f,[0.,0.5,1.],doyourworst=True)
12     print(f"iqi, bwe: {x}\n")
13
```



iqi, std: 0.6823278038280194

iqi, bwe: 0.6823278038280194

✓ 2 brents method

this hybrid method uses concepts of secant method, its generalizations and bisection. it expands dekkers method which uses secant backed up by bisection.

for continuous function f over bounded interval $[a, b]$ where $f(a) \cdot f(b) < 0$, brents method keeps track of current x_i that is best in sense of backward error and bracket $[a_i, b_i]$ of root. roughly speaking brents uses IQI to replace one of x_i, a_i, b_i if (1) the backward error improves and (2) the bracketing interval is cut at least in half. if that fails, the secant method is attempted. if that fails, bisection occurs which guarantees that uncertainty is at least halved.

✓ code, dekker

```

1  # https://blogs.mathworks.com/cleve/2015/10/12/zeroin-part-1-dekkers-algorithm/
2
3  import numpy as np
4  import scipy as sp
5
6  def dekker(f,a,b,display=0):...
68
69  def f(x):...
74
75  ab = (3,4)
76  #ab = (0,1); f = lambda x: pow(x,3) - pow(x,2) - x - 1
77
78  root_sys = sp.optimize.root(f,[19./6])
79  #print(f"scipy : {root_sys}\n")
80  dekker(f,ab[0],ab[1],display=8)
81
82

```



Show hidden output

✓ code, brent

```

1  # extends dekkers
2
3  import scipy as sp
4
5  f = lambda x: pow(x,3) - pow(x,2) - x - 1
6
7  def brent(f,a,b,tol=1e-8,max_iter=10):
8      c = a
9      fc = f(c)
10     fb = f(b)
11
12     i = 0
13     while (abs(c-b)>tol) and (i<max_iter): # fwe

```



```

scipy : 1.8392867552141612
dekker: 1.8392867464516185

```

✓ code, brent @scipy

```

1  import scipy as sp
2
3  f = lambda x: pow(x,3) - pow(x,2) - x - 1
4  ab = (0.,2.)
5
6  root,soln = sp.optimize.brentq(f,ab[0],ab[1],full_output=True)
7  print(f"root: {root}\n")
8  print(soln)
9  #print(soln.method) # sigh = n/a
10

```

⇌ root: 1.8392867552141612

```
    converged: True
      flag: converged
function_calls: 10
  iterations: 9
    root: 1.8392867552141612
```

- oscar veliz [@youtube](#) (multiple methods including brents method)
- [brents method](#)
- [richard brent](#)
- [theodorus dekker](#) with a [bonus](#)