

## ✓ 01.01 rootfinding : bisection

### ✓ 1 bracketing a root

#### ✓ definition 01 root

function  $f(x)$  has **root** at  $x = r$  if  $f(r) = 0$ .

#### ✓ theorem 02: bolzanos theorem

**intermediate value theorem, corollary 1.** if a continuous function has values of opposite sign inside an interval, then it has a root in that interval.<sup>[1]</sup>

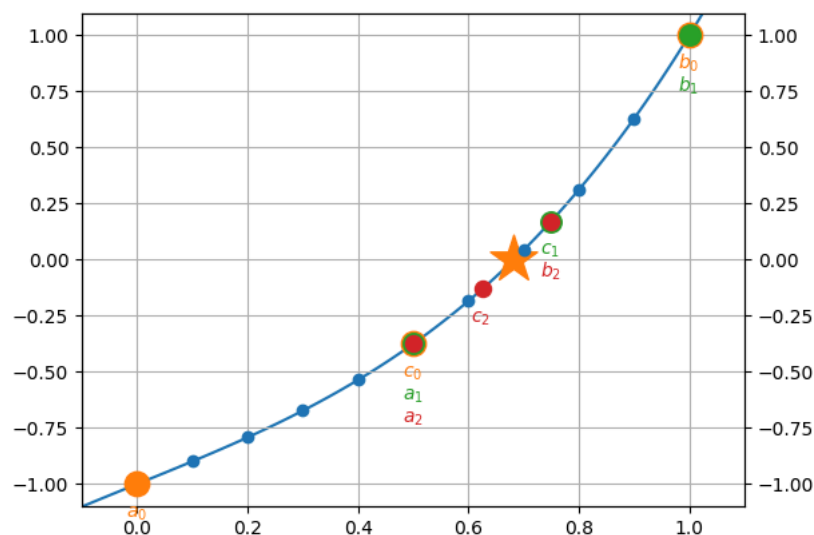
#### ✓ code, visual: bisection

```
1 # requires prior execution of bisect_expanded()
2
3 if __name__ == "__main__":
4
```

```
1 ani
```



bisection:  $x^3 + x - 1$ ,  $x \in [0, 1]$



☐ Once ☒ Loop ☐ Reflect

✓ algorithm: bisection

```
# given  $x \in [a,b]$  st  $f(a) \cdot f(b) < 0$ 
```

```
while (b-a)/2 > TOL
  c = (a+b)/2
  if f(c) = 0 then stop
  if f(a)·f(c) < 0 then
    b = c
  else
    a = c
  end
end
```

```
root_interval = [a,b]
root = (a+b)/2
```

✓ code, bisection

```
1 # algorithm, basic
2
3 def bisect(f,ab,tol):
4
5     a = ab[0]
6     b = ab[1]
7     while (b-a)/2 > tol:
8         c = (a+b)/2
9         fc = f(c)
10        if fc == 0:
11            break;
12        fa = f(a)
13        if fa*fc < 0:
14            b = c
15        else:
16            a = c
17
18    return c
19
```

```
1 # algorithm, expanded for lecture
2
3 def bisect_expanded(f,ab,tol,all=False,workspace=False):
4
```

✓ code, bisection, modify for ~~✗~~


```
1 # update in lecture to handle interval that does not contain root
```

## ✓ example 01

find a root of function  $f(x) = x^3 + x - 1$  on interval  $[0, 1]$  using bisection method.

## ✓ code, example 01

```
1 # basic, expanded, self-check
2
3 import scipy as sp
4 import textwrap
5
6 if __name__ == "__main__":
7
8     # problem
9     f = lambda x: pow(x,3) + x - 1.
10    ab = (0.,1.)
11    tol = 1e-08
12
13    # calc, basic
14    if True:
15        root = bisect(f,ab,tol)
16        print(f"[basic algorithm] root: {root} at tolerance {tol}.\n")
17
18    # calc, with details
19    if True:
20        root,ab_root,itors = bisect_expanded(f,ab,tol,all=True)
21        s_answer = f"[algorithm expanded for details] \
22            root {root} in final interval {ab_root} \
23            after {itors} iterations at tolerance {tol}."
24        print(textwrap.fill(" ".join(s_answer.split()),70),"\n")
25
26    # calc, with scipy
27    if True:
28        root,rr = sp.optimize.bisect(f,ab[0],ab[1],xtol=tol,rtol=tol,full_output=True)
29        iterations = rr.iterations # number of iterations # fyi
30        print(f"[scipy] root: {root} took {iterations} iterations at tolerance {tol}.\n")
31
```

 [basic algorithm] root: 0.6823277920484543 at tolerance 1e-08.

[algorithm expanded for details] root 0.6823277920484543 in final interval (0.6823277920484543, 0.6823278069496155) after 26 iterations at tolerance 1e-08.

[scipy] root: 0.6823277920484543 took 26 iterations at tolerance 1e-08.

## ✓ 2 speed and accuracy

if continuous  $f(x)$  and  $x \in [a, b]$ , then  $[a_n, b_n]$  of length  $\frac{b-a}{2^n}$  brackets the best solution after  $n$  steps. ie, solution  $r \approx x_c = \frac{a_n+b_n}{2}$  with

$$\begin{array}{ll} \text{error, bound:} & \Delta x < \epsilon \quad \Rightarrow \quad |x_c - x^*| < \frac{b-a}{2^{n+1}} \\ \text{function evaluations:} & n + 2. \end{array}$$

assess the efficiency of bisection by accuracy gained with each function evaluation. ie, there is one function evaluation per step and each step halves uncertainty.

note: sure, the previously mentioned rounding error limit applies but the operations required per  $f(x)$  are the same no matter how its usage within an approximation method. ie, its simpler to tally function calls.

## ✓ definition 03

a solution is **correct within  $p$  decimal places** if error is less than  $0.5 \times 10^{-p}$ .

## ✓ example 02

find root of  $f(x) = \cos x - x$  in interval  $[0, 1]$  within six correct places with bisection.

after  $n$  steps,

$$|x_c - r| < \frac{b-a}{2^{n+1}} \leq \frac{1 \times 10^{-6}}{2}$$

⇓

$$\epsilon = \frac{1-0}{2^{n+1}} \leq 0.5 \times 10^{-6}$$

⇓

$$n > \frac{6}{\log_{10} 2} \approx \frac{6}{0.301} \approx 19.9 \Rightarrow 20 \text{ steps required.}$$

## ✓ code, example 02

```
1 # example 02 modifies example 01
2
3 import scipy as sp
4 import textwrap
5
6 if __name__ == "__main__":
7
8     # problem
9     f = lambda x: np.cos(x) - x
10    ab = (0.,1.)
11    tol = 0.5e-06
12
13    # guess
14    n = 6/np.log10(2)
15    print(f"estimated steps: {n}.\n")
16
17    # calc, with details
18    if True:
19        root,ab_root,itors = bisect_expanded(f,ab,tol,all=True)
20        s_answer = f"[algorithm expanded for details] \
21        root {root} in final interval {ab_root} \
22        after {itors} iterations at tolerance {tol}."
23        print(textwrap.fill(" ".join(s_answer.split()),70),"\n")
24
25    # calc, with scipy
26    if True:
27        root,rr = sp.optimize.bisect(f,ab[0],ab[1],xtol=tol,rtol=tol,full_output=True)
28        iterations = rr.iterations # number of iterations # fyi
29        print(f"[scipy] root: {root} took {iterations} iterations at tolerance {tol}.\n")
30
31
```

➡ estimated steps: 19.931568569324174.

[algorithm expanded for details] root 0.7390851974487305 in final interval (0.7390842437744141, 0.7390851974487305) after 20 iterations at tolerance 5e-07.

[scipy] root: 0.7390847206115723 took 21 iterations at tolerance 5e-07.

## ✓ USW

error analysis helps set iteration limits, provides metrics when comparing efficiency (ie, accuracy gain per iteration).

## ✓ resources

- bisection [@scipy](#)

## ✓ references

[1] [bolzano, bernard](#). *intermediate value theorem (IVT), corollary one*, [1817](#). also: bolzano-weierstrass, [some fifty years later](#).