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.[1]

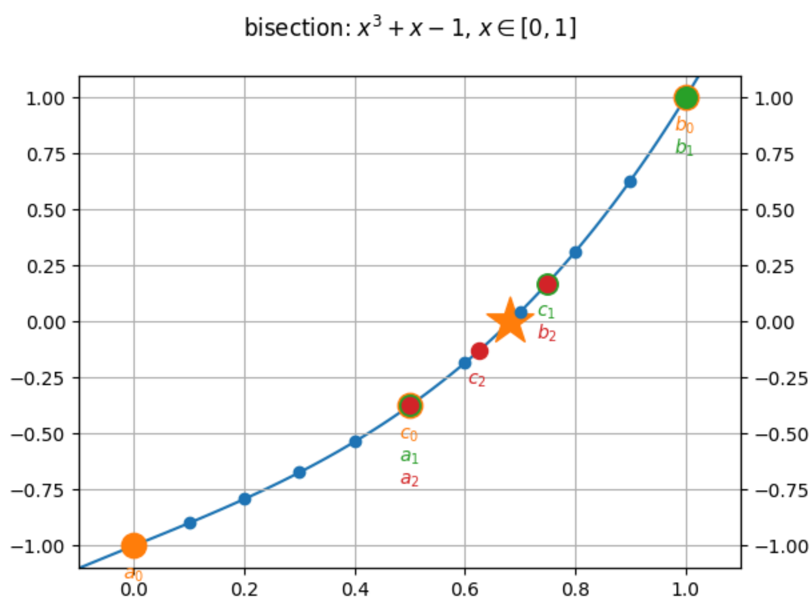
code, visual: bisection

```
# requires prior execution of bisect_expanded()

if __name__ == "__main__":

```

```
ani
```



Once Loop Reflect

algorithm: bisection

```
# given x ∈[a,b] st f(a)·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

```
# algorithm, basic

def bisect(f,ab,tol):

  a = ab[0]
  b = ab[1]
  while (b-a)/2 > tol:
    c = (a+b)/2
    fc = f(c)
    if fc == 0:
      break;
    fa = f(a)
    if fa*fc < 0:
      b = c
    else:
      a = c

  return c

```

```
# algorithm, expanded for lecture

def bisect_expanded(f,ab,tol,all=False,workspace=False):⋯
```

code, bisection, modify for $\nleq$

```
# 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

```python
# basic, expanded, self-check

import scipy as sp
import textwrap

if __name__ == "__main__":

  # problem
  f = lambda x: pow(x,3) + x - 1.
  ab = (0.,1.)
  tol = 1e-08

  # calc, basic
  if True:
    root = bisect(f,ab,tol)
    print(f"[basic algorithm] root: {root} at tolerance {tol}.\n")

  # calc, with details
  if True:
    root,ab_root,iters = bisect_expanded(f,ab,tol,all=True)
    s_answer = f"[algorithm expanded for details] \
      root {root} in final interval {ab_root} \
      after {iters} iterations at tolerance {tol}."
    print(textwrap.fill(" ".join(s_answer.split()),70),"\n")

  # calc, with scipy
  if True:
    root,rr = sp.optimize.bisect(f,ab[0],ab[1],xtol=tol,rtol=tol,full_output=True)
    iterations = rr.iterations # number of iterations # fyi
    print(f"[scipy] root: {root} took {iterations} iterations at tolerance {tol}.\n")

```

```
[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

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

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 $0.5x10^{-p}$.

example 02

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

after n steps,

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

$$\Downarrow$$

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

$$\Downarrow$$

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

code, example 02

```
# example 02 modifies example 01

import scipy as sp
import textwrap

if __name__ == "__main__":

  # problem
  f = lambda x: np.cos(x) - x
  ab = (0.,1.)
  tol = 0.5e-06

  # guess
  n = 6/np.log10(2)
  print(f"estimated steps: {n}.\n")

  # calc, with details
  if True:
    root,ab_root,iters = bisect_expanded(f,ab,tol,all=True)
    s_answer = f"[algorithm expanded for details] \
      root {root} in final interval {ab_root} \
      after {iters} iterations at tolerance {tol}."
    print(textwrap.fill(" ".join(s_answer.split()),70),"\n")

  # calc, with scipy
  if True:
    root,rr = sp.optimize.bisect(f,ab[0],ab[1],xtol=tol,rtol=tol,full_output=True)
    iterations = rr.iterations # number of iterations # fyi
    print(f"[scipy] root: {root} took {iterations} iterations at tolerance {tol}.\n")
```

```
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.