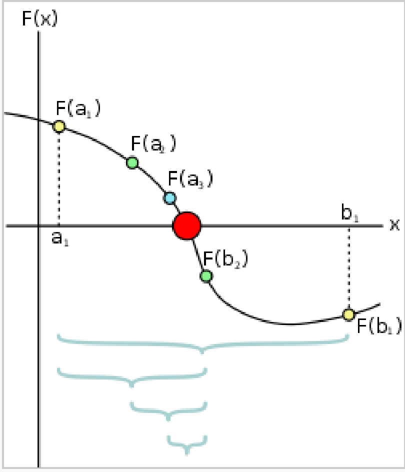
Bisection method

From Wikipedia, the free encyclopedia

This article is about searching continuous function values. For searching a finite sorted array, see binary search algorithm.

The **bisection method** in mathematics is a root-finding method that repeatedly bisects an interval and then selects a subinterval in which a root must lie for further processing. It is a very simple and robust method, but it is also relatively slow. Because of this, it is often used to obtain a rough approximation to a solution which is then used as a starting point for more rapidly converging methods.[1] The method is also called the **interval halving** method,[2] the **binary search method**,[3] or the **dichotomy method**.[4]



A few steps of the bisection method applied over the starting range [a_1;b_1]. The bigger red dot is the root of the function.

Contents [hide]

The method [edit]

The method is applicable for numerically solving the equation $f(x) = 0$ for the real variable x, where f is a continuous function defined on an interval $[a, b]$ and where $f(a)$ and $f(b)$ have opposite signs. In this case a and b are said to bracket a root since, by the intermediate value theorem, the continuous function f must have at least one root in the interval (a, b).

At each step the method divides the interval in two by computing the midpoint $c = (a+b) / 2$ of the interval and the value of the function $f(c)$ at that point. Unless c is itself a root (which is very unlikely, but possible) there are now only two possibilities: either $f(a)$ and $f(c)$ have opposite signs and bracket a root, or $f(c)$ and $f(b)$ have opposite signs and bracket a root.[5] The method selects the subinterval that is guaranteed to be a bracket as the new interval to be used in the next step. In this way an interval that contains a zero of f is reduced in width by 50% at each step. The process is continued until the interval is sufficiently small.

Explicitly, if $f(a)$ and $f(c)$ have opposite signs, then the method sets c as the new value for b, and if $f(b)$ and $f(c)$ have opposite signs then the method sets c as the new a. (If $f(c)=0$ then c may be taken as the solution and the process stops.) In both cases, the new $f(a)$ and $f(b)$ have opposite signs, so the method is applicable to this smaller interval.[6]

Iteration tasks [edit]

The input for the method is a continuous function f, an interval $[a, b]$, and the function values $f(a)$ and $f(b)$. The function values are of opposite sign (there is at least one zero crossing within the interval). Each iteration performs these steps:

1. Calculate c, the midpoint of the interval, $c = 0.5 * (a + b)$.
2. Calculate the function value at the midpoint, $f(c)$.
3. If convergence is satisfactory (that is, $a - c$ is sufficiently small, or $f(c)$ is sufficiently small), return c and stop iterating.
4. Examine the sign of $f(c)$ and replace either $(a, f(a))$ or $(b, f(b))$ with $(c, f(c))$ so that there is a zero crossing within the new interval.

When implementing the method on a computer, there can be problems with finite precision, so there are often additional convergence tests or limits to the number of iterations. Although *f* is continuous, finite precision may preclude a function value ever being zero. For $f(x) = x - \pi$, there will never be a finite representation of *x* that gives zero. Floating point representations also have limited precision, so at some point the midpoint of [*a*, *b*] will be either *a* or *b*.

Algorithm [edit]

The method may be written in pseudocode as follows:[7]

```
INPUT: Function f, endpoint values a, b, tolerance TOL, maximum iterations NMAX
CONDITIONS: a < b, either f(a) < 0 and f(b) > 0 or f(a) > 0 and f(b) < 0
OUTPUT: value which differs from a root of f(x)=0 by less than TOL

N ← 1
While N ≤ NMAX # limit iterations to prevent infinite loop
  c ← (a + b)/2 # new midpoint
  If f(c) = 0 or (b – a)/2 < TOL then # solution found
    Output(c)
    Stop
  EndIf
  N ← N + 1 # increment step counter
  If sign(f(c)) = sign(f(a)) then a ← c else b ← c # new interval
EndWhile
Output("Method failed.") # max number of steps exceeded
```

Example: Finding the root of a polynomial [edit]

Suppose that the bisection method is used to find a root of the polynomial

$$f(x) = x^3 - x - 2\,.$$

First, two numbers a and b have to be found such that $f(a)$ and $f(b)$ have opposite signs. For the above function, $a = 1$ and $b = 2$ satisfy this criterion, as

$$f(1) = (1)^3 - (1) - 2 = -2$$

and

$$f(2) = (2)^3 - (2) - 2 = +4\,.$$

Because the function is continuous, there must be a root within the interval [1, 2].

In the first iteration, the end points of the interval which brackets the root are $a_1 = 1$ and $b_1 = 2$, so the midpoint is

$$c_1 = \frac{2+1}{2} = 1.5$$

The function value at the midpoint is $f(c_1) = (1.5)^3 - (1.5) - 2 = -0.125$. Because $f(c_1)$ is negative, $a = 1$ is replaced with $a = 1.5$ for the next iteration to ensure that $f(a)$ and $f(b)$ have opposite signs. As this continues, the interval between a and b will become increasingly smaller, converging on the root of the function. See this happen in the table below.

Iteration	a_n	b_n	c_n	$f(c_n)$
1	1	2	1.5	−0.125
2	1.5	2	1.75	1.6093750
3	1.5	1.75	1.625	0.6660156
4	1.5	1.625	1.5625	0.2521973
5	1.5	1.5625	1.5312500	0.0591125
6	1.5	1.5312500	1.5156250	−0.0340538
7	1.5156250	1.5312500	1.5234375	0.0122504
8	1.5156250	1.5234375	1.5195313	−0.0109712
9	1.5195313	1.5234375	1.5214844	0.0006222
10	1.5195313	1.5214844	1.5205078	−0.0051789

11	1.5205078	1.5214844	1.5209961	−0.0022794
12	1.5209961	1.5214844	1.5212402	−0.0008289
13	1.5212402	1.5214844	1.5213623	−0.0001034
14	1.5213623	1.5214844	1.5214233	0.0002594
15	1.5213623	1.5214233	1.5213928	0.0000780

After 13 iterations, it becomes apparent that there is a convergence to about 1.521: a root for the polynomial.

Analysis [edit]

The method is guaranteed to converge to a root of *f* if *f* is a continuous function on the interval [*a*, *b*] and *f*(*a*) and *f*(*b*) have opposite signs. The absolute error is halved at each step so the method converges linearly, which is comparatively slow.

Specifically, if $c_1 = (a+b)/2$ is the midpoint of the initial interval, and c_n is the midpoint of the interval in the *n*th step, then the difference between c_n and a solution *c* is bounded by[8]

$$|c_n - c| \leq \frac{|b-a|}{2^n}.$$

This formula can be used to determine in advance the number of iterations that the bisection method would need to converge to a root to within a certain tolerance. The number of iterations needed, *n*, to achieve a given error (or tolerance), ε, is given by: $n = \log_2\left(\frac{\epsilon_0}{\epsilon}\right) = \frac{\log \epsilon_0 - \log \epsilon}{\log 2},$

where $\epsilon_0 = \text{initial bracket size} = b - a.$

Therefore, the linear convergence is expressed by $\epsilon_{n+1} = \text{constant} \times \epsilon_n^m, \;\; m = 1.$

See also [edit]

- Secant method
- Newton's method
- Root-finding algorithm
- Binary search algorithm
- Lehmer–Schur algorithm, generalization of the bisection method in the complex plane
- Nested intervals
- Brent's method

References [edit]

1. ^ Burden & Faires 1985, p. 31
2. ^ http://siber.cankaya.edu.tr/NumericalComputations/ceng375/node32.html
3. ^ Burden & Faires 1985, p. 28
4. ^ Encyclopedia of Mathematics
5. ^ If the function has the same sign at the endpoints of an interval, the endpoints may or may not bracket roots of the function.
6. ^ Burden & Faires 1985, p. 28 for section
7. ^ Burden & Faires 1985, p. 29. This version recomputes the function values at each iteration rather than carrying them to the next iterations.
8. ^ Burden & Faires 1985, p. 31, Theorem 2.1

- Burden, Richard L.; Faires, J. Douglas (1985), "2.1 The Bisection Algorithm", *Numerical Analysis* (3rd ed.), PWS Publishers, ISBN 0-87150-857-5

Further reading [edit]

- Corliss, George (1977), "Which root does the bisection algorithm find?", *SIAM Review* **19** (2): 325–327, doi:10.1137/1019044, ISSN 1095-7200
- Kaw, Autar; Kalu, Egwu (2008), *Numerical Methods with Applications* (1st ed.)

External links [edit]

- Weisstein, Eric W., "Bisection", *MathWorld*.
- Bisection Method Notes, PPT, Mathcad, Maple, Matlab, Mathematica from Holistic Numerical Methods Institute

Wikiversity has learning materials about ***The bisection method***

The Wikibook *Numerical Methods* has a page on the topic of: ***Equation Solving***

Categories: Root-finding algorithms