Numerical Analysis Series

Bisection Method

(Also known as Bolzano or Interval Halving Method)

Author
Shahad Uddin
shahaduddin.com

Date
February 2, 2026
Version 1.0

1. Theoretical Background

Mathematical Definition

Let $f(x)$ be a function that is continuous between a and b. For definiteness, let $f(a)$ be negative and $f(b)$ be positive (or vice versa).

According to the **Intermediate Value Theorem**, if $f(a) \cdot f(b) < 0$, then there exists at least one root of $f(x) = 0$ lying between a and b.

The Method Steps:

1. Find two points, a and b, such that $f(a)$ and $f(b)$ have opposite signs.
2. Calculate the first approximation (midpoint):

$$x_{new} = \frac{a + b}{2} \tag{1}$$

3. Check the sign of $f(x_{new})$:
   - If $f(x_{new}) = 0$, then x_{new} is the exact root.
   - If $f(a) \cdot f(x_{new}) < 0$, the root lies between a and x_{new}. Set $b = x_{new}$.
   - If $f(a) \cdot f(x_{new}) > 0$, the root lies between x_{new} and b. Set $a = x_{new}$.
4. Repeat the process until the interval is sufficiently small (within tolerance).

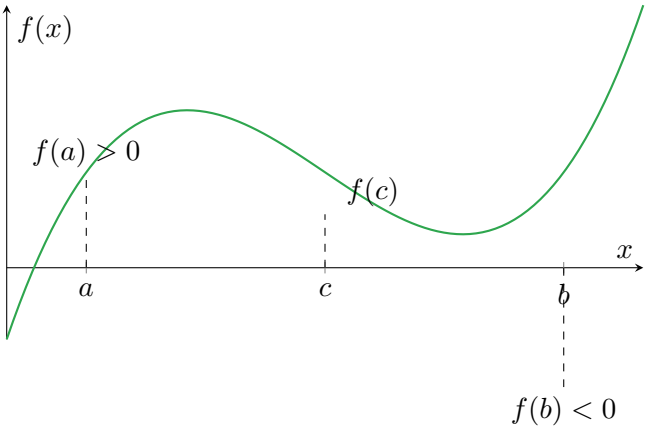

Fig 1. Visualizing the Interval Halving Process

2. Manual Calculation Example

Problem: Find a root of the equation $x^3 - x - 11 = 0$ correct to four decimals using the bisection method.

Solution: Let $f(x) = x^3 - x - 11$.

- $f(2) = 2^3 - 2 - 11 = 8 - 13 = -5$ (< 0)
- $f(3) = 3^3 - 3 - 11 = 27 - 14 = 13$ (> 0)

Since $f(2)$ is negative and $f(3)$ is positive, a root lies between 2 and 3.

Iteration 1:

$$x_1 = \frac{2+3}{2} = 2.5$$

$$f(2.5) = (2.5)^3 - 2.5 - 11 = 15.625 - 13.5 = 2.125 \quad (+ve)$$

Root lies between 2 and 2.5.

Iteration 2:

$$x_2 = \frac{2+2.5}{2} = 2.25$$

$$f(2.25) = (2.25)^3 - 2.25 - 11 = -1.8593 \quad (-ve)$$

Root lies between 2.25 and 2.5.

Proceeding in this manner, the root converges to approximately ***2.3736****.*

3. Code Implementations

Note: The code below demonstrates solving $x^3 - x - 2 = 0$ to show versatility.

>PythonImplementation *(.py)*

```python
def bisection(f, a, b, tol=1e-6, max_iter=100):
    """
    Standard Bisection Method for root finding.
    Guaranteed convergence for continuous functions.
    """
    if f(a) * f(b) >= 0:
        raise ValueError("f(a) and f(b) must have opposite signs")

    for i in range(max_iter):
        c = (a + b) / 2
        # Check if root is found or interval is small enough
        if abs(f(c)) < tol or (b - a) / 2 < tol:
            return c

        # Narrow the interval
        if f(c) * f(a) < 0:
```

```
            b = c
        else:
            a = c

    return (a + b) / 2

# Example Usage
import math
func = lambda x: x**3 - x - 2 # Transcendental or Algebraic
root = bisection(func, 1, 2)
print(f"Root: {root}")
```

>FortranImplementation (*.f90*)

```fortran
program bisection
    implicit none
    real :: a, b, c, fc, tol
    integer :: i

    a = 1.0
    b = 2.0
    tol = 1e-6

    if (f(a) * f(b) >= 0) stop "Root not bracketed"

    do i = 1, 100
        c = (a + b) / 2.0
        fc = f(c)

        if (abs(fc) < tol .or. (b - a)/2.0 < tol) exit

        if (fc * f(a) < 0) then
            b = c
        else
            a = c
        end if
    end do

    print *, "Root: ", c

contains
    real function f(x)
        real, intent(in) :: x
        f = x**3 - x - 2.0
    end function f
end program bisection
```

>C++Implementation (*.cpp*)

```cpp
#include <iostream>
#include <cmath>

double bisection(double (*f)(double), double a, double b, double tol = 1e
    -6) {
    if (f(a) * f(b) >= 0) return 0; // Error handling

    double c;
    while ((b - a) / 2.0 > tol) {
        c = (a + b) / 2.0;
        if (std::abs(f(c)) < 1e-12) break;

        if (f(a) * f(c) < 0)
            b = c;
        else
            a = c;
    }
    return c;
}

```

```cpp
int main() {
    auto f = [](double x) { return std::pow(x, 3) - x - 2; };
    std::cout << "Root: " << bisection(f, 1, 2) << std::endl;
    return 0;
}
```