Daffodil International University

Group Assignment on Bisection Method

Course Code: CSE226

SUBMITTED BY

NAME: Hossain Mohammad Jayed

STUDENT ID: 0242310005101640

DEPARTMENT: Computer Science & Engineering (CSE)

SECTION: 64_G

SUBMITTED TO

NAME: MD. YOUSUF ALI

DEPARTMENT: Lecturer in Mathematics Department of CSE

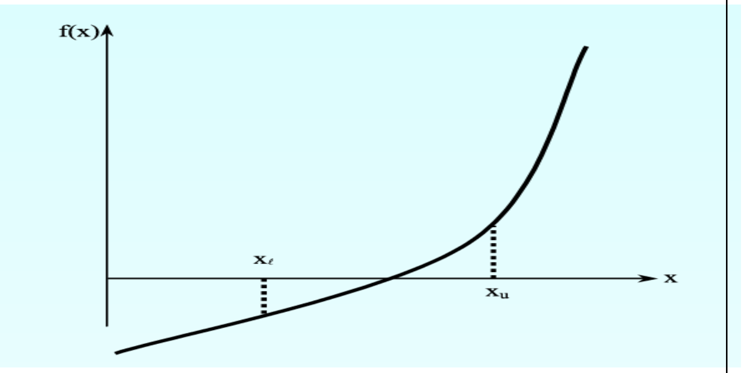
Date of submission: 30.11.2024

CSE226, Numerical Method

Assignment (Bisection Method)

Derivation of the Bisection Method

The Bisection Method is a simple and robust numerical technique used to find the root of a continuous function $f(x) = 0$. Within a specified interval $[x_l, x_u]$. This method works by repeatedly narrowing down the interval where the root lies, based on the behaviour of the function.



Derivation:

An equation $f(x) = 0$, where $f(x)$ is a real continuous function, has at least one root between x_l and x_u if $f(x_l)\,f(x_u) < 0$.

If the function is real, continuous, and changes sign, at least one root exists between the two points.

Algorithm of the Bisection Method

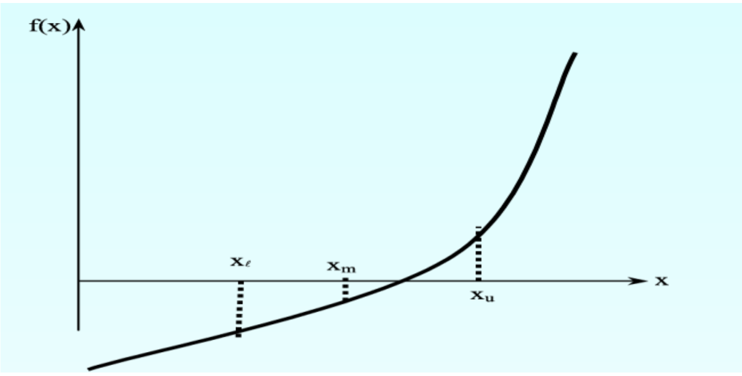
- **Objective:** To find an approximate root x of a continuous function f(x) within a given interval [a, b], where f(a)·f(b)<0 (i.e., the function changes sign over the interval).

- **Step-by-step Explanation:**

  - **Input Requirements:**
    - A continuous function $f(x)$.
    - Initial interval endpoints a and b such that $f(a) \cdot f(b) < 0$ (i.e., there is a root within the interval).
    - A tolerance value ϵ\epsilonϵ for determining the stopping condition (e.g., 10^{-6} for high precision).

  - **Initial Check:**
    - Ensure that $f(a) \cdot f(b) < 0$. If this condition is not met, the algorithm should return an error indicating that the chosen interval does not bracket a root.

  - **Iterative Procedure:**
    - **Step 1**: Calculate the midpoint c of the interval: $c = \frac{(a+b)}{2}$



    - **Step 2**: Evaluate $f(c)$:
      - If $f(c) = 0$ $(or\ \ |f(c)| < \ \epsilon)$, then c is the root, and the process ends.

- **Step 3**: Determine the subinterval containing the root:
  - If $f(a) \cdot f(c) < 0$, the root lies between a and c. Set b=c to update the interval.
  - If $f(c) \cdot f(b) < 0$, the root lies between c and b. Set a=c to update the interval.
- **Step 4**: Check the termination condition:
  - If $| b - a | < \epsilon$, the algorithm stops, and c is returned as the approximate root.

➢ **Loop Until Convergence:**

- Repeat **Step 1** to **Step 4** iteratively until the interval width |b−a| becomes smaller than the tolerance ϵ\epsilonϵ, ensuring that the solution has sufficient accuracy.

❖ **Pseudocode Representation:**

```
INPUT: f(x), a, b, tolerance ε

IF f(a) * f(b) >= 0 THEN
    PRINT "Invalid interval. No root found."
    EXIT
END IF


REPEAT
    c = (a + b) / 2
    IF f(c) == 0 OR |f(c)| < ε THEN
        PRINT "Root found at", c
        EXIT
    END IF
```

```
    IF f(a) * f(c) < 0 THEN
        b = c
    ELSE
        a = c
    END IF
UNTIL |b - a| < ε

PRINT "Approximate root at", c
```

❖ **Detailed Explanation of Each Step:**

- **Initial Check:** Ensuring $f(a) \cdot f(b) < 0$ verifies that there is a sign change over the interval [a, b], implying the presence of a root. Without this condition, the method cannot proceed.
- **Midpoint Calculation:** The midpoint c is the candidate for the root in each iteration. This helps divide the interval into two subintervals to narrow down the search.
- **Evaluation and Decision:**
  - If f(c) is zero or close to zero (within ϵ), ccc is considered the root, and the search ends.
  - The product $f(a) \cdot f(c) < 0$ implies that the root lies in the left subinterval [a, c], so b is updated to c.
  - If $f(c) \cdot f(b) < 0$, the root lies in the right subinterval [c, b], and a is updated to c.
- **Termination Condition:** When $|b-a| < \epsilon$, it indicates that the interval is sufficiently small, and ccc can be accepted as the approximate root with the desired accuracy.

❖ Key Characteristics:

- **Convergence:** The Bisection Method converges linearly, ensuring that with each iteration, the interval width is halved, leading to a progressively more accurate estimate of the root.

- **Robustness:** The method is reliable and guarantees convergence if the initial conditions are met.

- **Simplicity:** The method is straightforward to implement and understand, making it a popular choice for root-finding problems.

➢ Mathematical Analysis Table

Given Equation: $f(a) = x^2 - 4\ [0, 3]$ and tolerance: 0.001

Initial Values:

$$f(0) = -4 < 0, \qquad f(3) = 5 > 0$$

- **Table of Calculations:**

a	$f(a)$	b	$f(b)$	$c = \frac{a+b}{2}$	$f(c)$
0	-4	3	5	1.5	-1.75
1.5	-1.75	3	5	2.25	1.0625
1.5	-1.75	2.25	1.0625	1.875	-0.4844
1.875	-0.4844	2.25	1.0625	2.0625	0.2540
1.875	-0.4844	2.0625	0.2540	1.9688	-0.1240
1.9688	-0.1240	2.0625	0.2540	2.0175	0.0628
1.9688	-0.1240	2.0157	0.0628	1.9923	-0.0309
1.9923	-0.0309	2.0157	0.0628	2.004	0.0160
1.9923	-0.0309	2.004	0.0160	1.998 or 2	-0.0074

$\boldsymbol{Answer}: 2$

Coding of Bisection Method in C

Here's a simple C code to implement the Bisection Method:

```c
#include <stdio.h>

#include <math.h>


// Define the function f(x). You can change this to any continuous function

double f(double x)

{

   return x*x - 4;  // Example: f(x) = x^2 - 4, root at x = 2 and x = -2

}


// Bisection method implementation

double bisection(double a, double b, double epsilon, int max_iter)

{

   double c;

   int iter = 0;


   // Check if the initial values satisfy the condition f(a) * f(b) < 0

   if (f(a) * f(b) > 0) {

      printf("No root in this interval [%.2f, %.2f].\n", a, b);

      return -1;  // Indicating no root found

   }


   // Bisection method loop

   while ((b - a) / 2.0 > epsilon && iter < max_iter)

{

      // Find the midpoint

      c = (a + b) / 2.0;
```

```
        // If f(c) is close to zero, return c as the root
        if (f(c) == 0.0) {
            printf("Root found at c = %.5f\n", c);
            return c;
        }

        // Update the interval
        if (f(a) * f(c) < 0)
{
            b = c;  // Root is in the left half
        }
else
{
            a = c;  // Root is in the right half
        }

        iter++;
    }

    // Return the best approximation of the root
    c = (a + b) / 2.0;
    printf("\n\nRoot approximation after %d iterations: c = %.5f\n", iter, c);
    return c;
}

int main()
```

```
{
    double a, b, epsilon;
    int max_iter;

    // Input for the interval [a, b], tolerance, and max iterations
    printf("Enter the interval [a, b] (Example: 0 3): ");
    scanf("%lf %lf", &a, &b);

    printf("Enter the tolerance (Example: 0.001): ");
    scanf("%lf", &epsilon);

    printf("Enter the maximum number of iterations (Example: 50): ");
    scanf("%d", &max_iter);

    // Call the bisection method
    double root = bisection(a, b, epsilon, max_iter);

    if (root != -1)
{
        printf("The approximate root is: %.5f\n", root);
    }

    return 0;
}
```

- ✓ **Output:**

Output

```
Enter the interval [a, b] (Example: 0 3): 0 3
Enter the tolerance (Example: 0.001): 0.001
Enter the maximum number of iterations (Example: 50): 69


Root approximation after 11 iterations: c = 2.00024
The approximate root is: 2.00024


=== Code Execution Successful ===
```

➢ Explanation of the Code:

- $f(double\ x)$: Function definition that takes xxx and returns f(x). You can modify it based on your problem.
- $bisection(double\ a, double\ b, double\ tol)$: The core function that applies the Bisection Method.
- $main()$: Takes user input for the interval endpoints aaa and bbb, and the desired tolerance ϵ. It then calls the bisection() function.