Numerical Method Lab

1. To find the roots of non-linear equation using Bisection method.

Solution:

Algorithm of Bisection Method for finding root:

1. **Input**: Function $f(x)$, interval $[a, b]$, and a tolerance value.
2. Check if $f(a) \times f(b) \geq 0$. If true, stop: no root exists within this interval.
3. Set $c = \frac{a+b}{2}$.
4. While $|b - a| \geq \text{tolerance}$:
   a. Evaluate $f(c)$.
   b. If $f(c) = 0$, then c is the root. Stop the process.
   c. If $f(c) \times f(a) < 0$, set $b = c$.
   d. If $f(c) \times f(b) < 0$, set $a = c$.
   e. Update $c = \frac{a+b}{2}$.
5. Output the final value of c as the approximate root.

Suppose we have a function: $f(x) = 3x - cos(x) - 1$

Now we need a and b. [0,1]
এখানে a এবং b এর মান নেয়ার সময় একটা কন্ডিশন মাথায় রাখতে হবে। তা হলোঃ $f(a) * f(b) < 0;$

এখানে আমরা a এবং b এর জন্য এমন মান নিবো যাতে ২টা ফাংশন গুন করলে ০ এর ছোট হয়।
এ জন্য আমরা [0,1] এটা না হলে [1,2] এটা না হলে [2,3] এভাবে চলতে থাকবে I তাও না হলে মাইনেস মান দিয়েও আমরা চেক করে দেখবো।

$a = 0 \qquad f(a) = f(0) = 3 * 0 - cos\,0 - 1 = -2$

$b = 1 \qquad f(b) = f(1) = 3 * 1 - cos\,1 - 1 = 1.46$

$\therefore f(a) * f(b) < 0$

$\Rightarrow -2 * 1.46$

$\Rightarrow -2.92$ [Note: এখানে আমরা দেখতে পারছি শর্ত মেনেছে তাই আমরা ধরে নইতে পারি আমাদের রুট ০ আর ১ এর মাঝে আছে]

Let's Find the root:

Befor jump we need to know 1 thing:

$If\ f(a) * f(c) = positive\ value\ then\ a = c;$

$If\ f(a) * f(c) = negative\ value\ then\ b = c;$

Now Lets go:

a	b	$f(a)$	$f(b)$	$c = \frac{a+b}{2}$	$f(c)$
0	1	-2	1.4597	0.5	-0.377583
0.5	1	-0.377583	1.4597	0.75	0.518311
0.5	0.75	-0.377583	0.518311	0.625	0.0640369
0.5	0.625	-0.377583	0.0640369	0.5625	-0.158424
0.5625	0.625	-0.158424	0.0640369	0.59375	-0.0475985
0.59375	0.625	-0.0475985	0.0640369	0.609375	0.0081191
0.59375	0.609375	-0.0475985	0.0081191	0.601562	-0.0197649
0.601562	0.609375	-0.0197649	0.0081191	0.605469	-0.00582915
0.605469	0.609375	-0.00582915	0.0081191	0.607422	0.00114341
0.605469	0.607422	-0.00582915	0.00114341	0.606445	-0.00234326

Solve with iteration :

```
#include <bits/stdc++.h>

using namespace std;

double equation(double x) {

    // Define your equation here

    // For example, let's solve 3*x - cos(x)  - 1

    return 3*x - cos(x)  - 1;

}


double bisectionMethod(double a, double b, double tolerance) {
```

```
    double c;

int n=1;

    while (fabs(b - a    ) >= tolerance) {




        c = (a + b) / 2;

cout<<"Iteration: "<<n<< " a = " << a << " b = " << b << " f(a) "<<equation(a)<< " f(b) "<<equation(b)<<" c = "<<c<<" f(c) "<<equation(c)<<endl;



        if (equation(c) == 0.0)

            return c;


        if (equation(c) * equation(a) < 0)

            b = c;

        else

            a = c;

n++;

    }

    cout<<"Iteration: "<<n<< " a = " << a << " b = " << b << " f(a) "<<equation(a)<< " f(b) "<<equation(b)<<" c = "<<c<<" f(c) "<<equation(c)<<endl;



    return c;

}
```

```
int main() {
    double a, b, tolerance;

    cout << "Enter the interval [a, b]: ";
    cin >> a >> b;

    cout << "Enter the tolerance: ";
    cin >> tolerance;

    double root = bisectionMethod(a, b, tolerance);

    cout << "Approximate root: " << root << endl;

    return 0;
}
```

Additional info:

The condition `while (fabs(b - a) >= tolerance)` in the code ensures that the bisection method keeps running until the interval between the two values (let's call them `a` and `b`) becomes smaller than the desired level of accuracy, which is defined as `tolerance`.

Imagine you're trying to find where a number is on a line, but you can only see a range on that line (from `a` to `b`). To determine the number more precisely, you need to keep reducing the range until it's very small. The `while` condition does just that – it keeps the method running until the range (the difference between `a` and `b`) is tinier than what you consider acceptable (tolerance). This helps to pinpoint the location of the number you're seeking.

So, the smaller the `tolerance`, the more precise the final result will be, because it forces the method to keep refining the range until it's very, very small, giving a more accurate approximation of the number you're looking for.