**Assignment Number**

**Problem Statement**

Program in C to find the root of a transcendental equation using Bisection Method.

**Theory**

The bisection method in mathematics is a root-finding method that repeatedly [bisects](https://en.wikipedia.org/wiki/Bisection) an [interval](https://en.wikipedia.org/wiki/Interval_(mathematics)) and then selects a subinterval in which a [root](https://en.wikipedia.org/wiki/Root_of_a_function) 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.

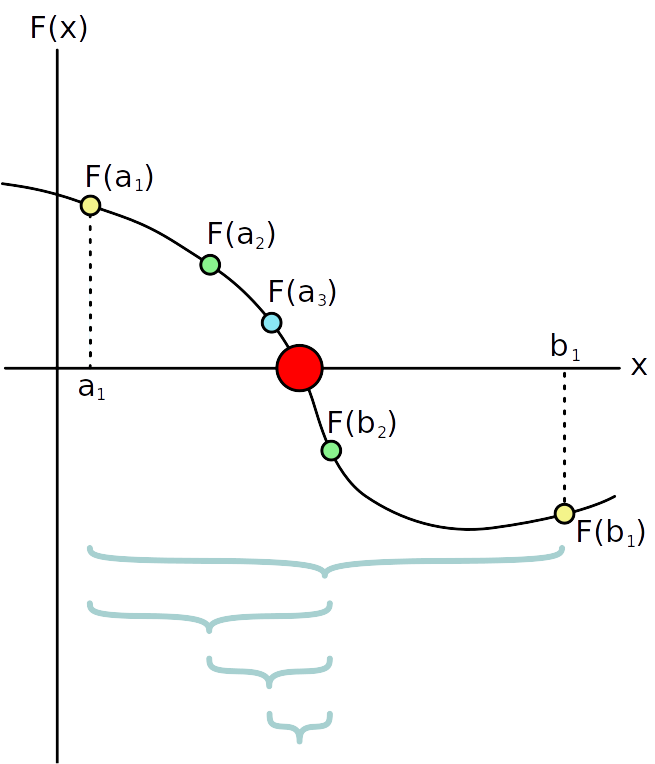
The method is applicable for numerically solving the equation *f*(*x*) = 0 for the [real](https://en.wikipedia.org/wiki/Real_number) variable *x*, where *f* is a continuous function defined on an interval [*a*, *b*] and where *f*(*a*) and *f*(*b*) have opposite signs. In this case *a a*nd *b* are said 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. 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.

**Iteration tasks**

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* = *a* + *b*/2.
2. Calculate the function value at the midpoint, *f*(*c*).
3. If convergence is satisfactory (that is, *c –* *a* 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.

**Algorithm**

**Input :**

1. y = f(x), the function to find root of
2. The lower bound of the root, say a
3. The upper bound of the root, say b
4. A predefined small quantity, say EPSILON, which will denote the proximity of two roots to be considered as equal
5. A predefined number of steps to continue the iteration for, say STEPS

**Output :** The root of the function, say ROOT

**Steps :**

1. At first, y = f(x) is defined
2. Input a
3. Input b
4. If(f(b) \* f(a) > 0)

Then

* 1. Goto step 2

1. prevroot = a
2. c = ( b + a ) / 2
3. If(c – prevroot < EPSILON)

Then

* 1. ROOT = c
  2. Print “Root found : “, ROOT
  3. Exit

1. If(f(b) \* f(c) < 0)

Then

* 1. b = c

1. Else
   1. a = c
2. prevroot = c
3. STEPS = STEPS – 1
4. If(STEPS > 0)

Then

* + 1. Goto step 6

1. Else
   * 1. Print “Root does not converge in given steps!”
     2. Exit
2. End

**Source Code**

#include <stdio.h>

#include <math.h>

#define f(x) (x\*x\*x - 3\*x\*x + 3\*x – 1) // the equation is to be defined here

#define EPSILON 0.00000001

#define STEPS 100

int main(){

double a, b;

printf("\nEquation : x^3 - 3\*x^2 + 3\*x - 1");

restart:

printf("\nEnter initial approximation of the root : ");

scanf("%lf%lf", &a, &b);

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

printf("The root does not lie between %g and %g!", a, b);

goto restart;

}

int it = 0;

double prevroot = a, preva = a, prevb = b;

printf("\nIteration\t a \t f(a) \t b \t f(b) \t c \t f(c) ");

printf("\n=========\t=========\t=========\t=========\t=========\t=========\t=========");

while(it < STEPS){

double c = (a+b)/2;

printf("\n%9d\t", (it+1));

if(preva != a)

printf("\*%8lf", a);

else

printf("%9lf", a);

printf("\t%9lf\t", f(a));

if(prevb != b)

printf("\*%8lf", b);

else

printf("%9lf", b);

printf("\t%9lf\t%9lf\t%9lf", f(b), c, f(c));

preva = a; prevb = b;

if(fabs(c-prevroot) < EPSILON){

printf("\nRoot found : %10lf", prevroot);

return 0;

}

if(f(b) \* f(c) < 0){

a = c;

}

else{

b = c;

}

prevroot = c;

it++;

}

printf("\nRoot does not converge in %2d steps!", it);

}

**Input and Output**

**Set 1:**

Equation : x^3 - 5\*x + 3

Enter initial approximation of the root : 1 0

Iteration a f(a) b f(b) c f(c)

======== ======= ======== ======== ======== ======= ========

1 1.000000 0.000000 0.000000 -1.000000 0.500000 -0.125000

2 1.000000 0.000000 \*0.500000 -0.125000 0.750000 -0.015625

3 1.000000 0.000000 \*0.750000 -0.015625 0.875000 -0.001953

4 1.000000 0.000000 \*0.875000 -0.001953 0.937500 -0.000244

5 1.000000 0.000000 \*0.937500 -0.000244 0.968750 -0.000031

6 1.000000 0.000000 \*0.968750 -0.000031 0.984375 -0.000004

7 1.000000 0.000000 \*0.984375 -0.000004 0.992188 -0.000000

8 1.000000 0.000000 \*0.992188 -0.000000 0.996094 -0.000000

9 1.000000 0.000000 \*0.996094 -0.000000 0.998047 -0.000000

10 1.000000 0.000000 \*0.998047 -0.000000 0.999023 -0.000000

11 1.000000 0.000000 \*0.999023 -0.000000 0.999512 -0.000000

12 1.000000 0.000000 \*0.999512 -0.000000 0.999756 -0.000000

13 1.000000 0.000000 \*0.999756 -0.000000 0.999878 -0.000000

14 1.000000 0.000000 \*0.999878 -0.000000 0.999939 -0.000000

15 1.000000 0.000000 \*0.999939 -0.000000 0.999969 -0.000000

16 1.000000 0.000000 \*0.999969 -0.000000 0.999985 -0.000000

17 1.000000 0.000000 \*0.999985 -0.000000 0.999992 -0.000000

18 1.000000 0.000000 \*0.999992 -0.000000 0.999996 0.000000

19 1.000000 0.000000 \*0.999996 0.000000 0.999998 0.000000

20 1.000000 0.000000 \*0.999998 0.000000 0.999999 0.000000

21 1.000000 0.000000 \*0.999999 0.000000 1.000000 0.000000

22 1.000000 0.000000 \*1.000000 0.000000 1.000000 0.000000

23 1.000000 0.000000 \*1.000000 0.000000 1.000000 0.000000

24 1.000000 0.000000 \*1.000000 0.000000 1.000000 0.000000

25 1.000000 0.000000 \*1.000000 0.000000 1.000000 0.000000

Root found : 1.000000

**Set 2 :**

Equation : x^3 - 5\*x + 3

Enter initial approximation of the root : 0 1

Iteration a f(a) b f(b) c f(c)

======== ========= ======== ======== ======== ======== ========

1 0.000000 3.000000 1.000000 -1.000000 0.500000 0.625000

2 \*0.500000 0.625000 1.000000 -1.000000 0.750000 -0.328125

3 0.500000 0.625000 \*0.750000 -0.328125 0.625000 0.119141

4 \*0.625000 0.119141 0.750000 -0.328125 0.687500 -0.112549

5 0.625000 0.119141 \*0.687500 -0.112549 0.656250 0.001373

6 \*0.656250 0.001373 0.687500 -0.112549 0.671875 -0.056080

7 0.656250 0.001373 \*0.671875 -0.056080 0.664062 -0.027475

8 0.656250 0.001373 \*0.664062 -0.027475 0.660156 -0.013081

9 0.656250 0.001373 \*0.660156 -0.013081 0.658203 -0.005861

10 0.656250 0.001373 \*0.658203 -0.005861 0.657227 -0.002246

11 0.656250 0.001373 \*0.657227 -0.002246 0.656738 -0.000437

12 0.656250 0.001373 \*0.656738 -0.000437 0.656494 0.000468

13 \*0.656494 0.000468 0.656738 -0.000437 0.656616 0.000016

14 \*0.656616 0.000016 0.656738 -0.000437 0.656677 -0.000211

15 0.656616 0.000016 \*0.656677 -0.000211 0.656647 -0.000097

16 0.656616 0.000016 \*0.656647 -0.000097 0.656631 -0.000041

17 0.656616 0.000016 \*0.656631 -0.000041 0.656624 -0.000013

18 0.656616 0.000016 \*0.656624 -0.000013 0.656620 0.000002

19 \*0.656620 0.000002 0.656624 -0.000013 0.656622 -0.000006

20 0.656620 0.000002 \*0.656622 -0.000006 0.656621 -0.000002

21 0.656620 0.000002 \*0.656621 -0.000002 0.656621 -0.000000

22 0.656620 0.000002 \*0.656621 -0.000000 0.656620 0.000001

23 \*0.656620 0.000001 0.656621 -0.000000 0.656620 0.000000

24 \*0.656620 0.000000 0.656621 -0.000000 0.656620 -0.000000

25 0.656620 0.000000 \*0.656620 -0.000000 0.656620 0.000000

26 \*0.656620 0.000000 0.656620 -0.000000 0.656620 0.000000

27 \*0.656620 0.000000 0.656620 -0.000000 0.656620 -0.000000

Root found : 0.656620

**Discussion**

1. Bisection method is the safest and it always converges. The bisection method is the simplest of all other methods and is guaranteed to converge for a continuous function.
2. It is always possible to find the number of steps required for a given accuracy and the new methods can also be developed from bisection method and bisection method plays a very crucial role in computer science research.
3. However, bisection method is less accurate than its companions, hence it is less used where precision plays a major role.