INTERNATIONAL JOURNAL OF PROGRESSIVE RESEARCH IN ENGINEERING MANAGEMENT AND SCIENCE (IJPREMS)

www.ijprems.com
editor@ijprems.com

Vol. 03, Issue 09, September 2023, pp : 238-239

e-ISSN : 2583-1062

Impact Factor : 5.725

A ROBUST BISECTION METHOD FOR ROOT FINDING IN C

Dr. Jyothi. G[1], Dr. M. Dhanashmi[2], K. Bhanu Priya[3], P. Sarayu[4], T. Sai Durga[5], N. Keerthi[6], Sk. Nasrin[7], K. Divya[8]

[1,2,3]Lecturers, Department of Mathematics, Sri Durga Malleswara Siddhartha Mahila Kalasala, Vijayawada, A.P, India.

[4,5,6,7,8]Students, Mathematics, Sri Durga Malleswara Siddhartha Mahila Kalasala, Vijayawada, A.P, India.

DOI : https://www.doi.org/10.56726/IRJMETS32045

ABSTRACT

This is also an iterative method. To find root repeatedly bisect an interval and then selects a subinterval in which a root must lie for further processing. Algorithm is quite simple and robust, only requirement is that initial search interval must encapsulates the actual root.

1. INTRODUCTION

Bisection method is a simple iteration method to solve equation. This method is also known as Bolzano method of successive bisection. Some times it is referred to as half interval method. Suppose we know an equation of the form f(x)=0 has exactly one real root between two real numbers x_0, x_1. The number is chosen such that $f(x_0) and\ f(x_1)$ will have opposite signs. Let us bisect the interval $[x_0, x_1]$ into two half intervals and find the mid point $x_2 = \frac{x_0+x_1}{2}$. If $f(x_2) = 0$ then x_2 is a root. If $f(x_1) and\ f(x_2)$ have same sign then the root lies between $x_0\ and\ x_2$. The interval is taken as $[x_0, x_2]$. Otherwise the root lies in the interval $[x_2, x_1]$. Repeating the process of bisection we obtain successive sub intervals which are smaller. At each iteration, we get the mid point as a better approximation of the root. This process is terminated when interval is smaller than the desired accuracy. This is also called as "Interval Halving Method". Given a function f(x) continuous on an interval [a,b] and f(a).f(b)<0

Do $c = \frac{(a+b)}{2}$

If f(a)*f(c)<0 then b=c

Else a=c

While (none of the convergence criteria $c_1, c_2\ or\ c_3$ is satisfied)

Where the criteria for convergence are

c_1 : Fixing a priori the total number of bisection iterations N i.e, the length of the interval or the maximum error after N iterations in this case is less than $\frac{|b-a|}{2N}$.

c_2 : By testing the condition $|c_i - c_{i-1}|$ less than some tolerance limit, say epsilon, fixed threshold.

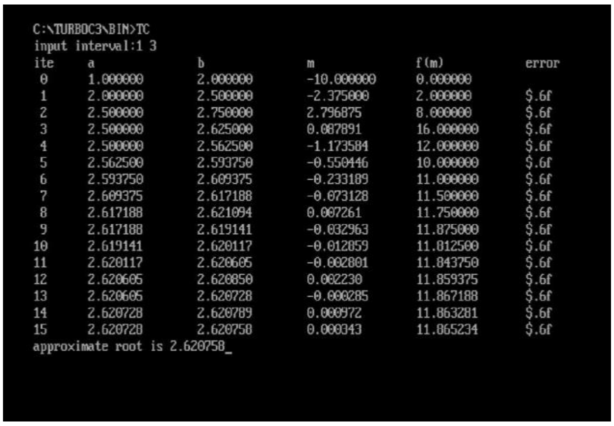
c_3 : By testing the condition $|f(c_i)|$ less than some tolerance limit alpha again fixed threshold.

2. C IMPLEMENTATION

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define f(x)((x*x*x)-18)
int main()
float a=0,b=0, error=0,m, mold;
printf("Input Interval:");
int i=0;
scanf("%f %f",&a,&b);
if((f(a)*f(b))>0)
{
printf("Invalid Interval Exit!");
exit(1);
}
else if(f(a)==0||f(b)==0)
```

```
{
printf("Root is one of interval bounds exit(0) root is %f \n",f(0)==0?a:b);
}
printf("Ite\ta\t\tb\t\tm\t\tf(m)\t\terror\n");
do
{
mold=m;
m=(a+b)/2;
printf("%2d\t%4.6f\t%4.6f\t%4.6f\t%4.6f\t",i++,a,b,m,f((m)));
if(f(m)==0)
{
printf("\n Root is %4.6f",m);
}
else if ((f(a)*f(m))<0)
{
b=m;
}
else a=m;
error=fabs (m-mold);
if(i==1)
{
printf("\n");
}
else
printf("%4.6f\n",error);
}
while(error>0.00005);
printf("Approximate Root is %4.6f",m);
return 0;
```

3. OUTPUT



C:\TURBOC3\BIN>TC
input interval:1 3

ite	a	b	m	f(m)	error
0	1.000000	2.000000	-10.000000	0.000000	
1	2.000000	2.500000	-2.375000	2.000000	$.6f
2	2.500000	2.750000	2.796875	8.000000	$.6f
3	2.500000	2.625000	0.087891	16.000000	$.6f
4	2.500000	2.562500	-1.173584	12.000000	$.6f
5	2.562500	2.593750	-0.550446	10.000000	$.6f
6	2.593750	2.609375	-0.233189	11.000000	$.6f
7	2.609375	2.617188	-0.073128	11.500000	$.6f
8	2.617188	2.621094	0.007261	11.750000	$.6f
9	2.617188	2.619141	-0.032963	11.875000	$.6f
10	2.619141	2.620117	-0.012859	11.812500	$.6f
11	2.620117	2.620605	-0.002801	11.843750	$.6f
12	2.620605	2.620850	0.002230	11.859375	$.6f
13	2.620605	2.620728	-0.000285	11.867188	$.6f
14	2.620728	2.620789	0.000972	11.863281	$.6f
15	2.620728	2.620758	0.000343	11.865234	$.6f

approximate root is 2.620758_

4. CONCLUSION

Overall, the paper provides a clear explanation of the bisection method and offers a practical implementation in C, showcasing its effectiveness in finding approximate roots of equations within specified intervals.

5. REFERENCES

[1] https://www.codewithc.com
[2] Introduction to Numerical Analysis with C programs by A Mate-2002
[3] C Programming and Numerical Analysis.