**Assignment no.**

**Problem Statement: PROGRAM IN "C" TO DETERMINE THE ROOTS (CORRECT UPTO 5TH DECIMAL PLACES)**

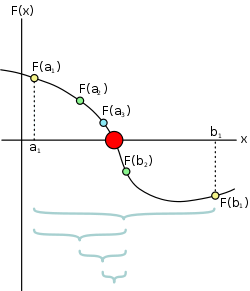
**OF THE FOLLOWING EQUATION BY USING "Bisection Method".**

**f(x) = x^3 - 2\*x – 3**

**Theory:**

The method is applicable for numerically solving the equation f(x) = 0 for the real 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 and 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.[5] 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



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:

Calculate c, the midpoint of the interval, c = a + b/2.

Calculate the function value at the midpoint, f(c).

If convergence is satisfactory (that is, c - a is sufficiently small, or |f(c)| is sufficiently small), return c and stop iterating. 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.

When implementing the method on a computer, there can be problems with finite precision, so there are often additional convergence tests or limits to the number of iterations. Although f is continuous, finite precision may preclude a function value ever being zero. For example, consider f(x) = x − π; there will never be a finite representation of x that gives zero. Additionally, the difference between a and b is limited by the floating point precision; i.e., as the difference between a and b decreases, at some point the midpoint of [a, b] will be numerically identical to (within floating point precision of) either a or b.

**Variable Listing:**

|  |  |  |
| --- | --- | --- |
| **Variable Name** | **Data Type** | **Purpose** |
| a, b | float | For storing intervals of a given function |
| c | float | Stores the formula of bisection |
| c\_prev | float | Stores the previous value of c |
| error | float | Stores the errors of c and c\_prev |
| i | integer | Loop variable |

**Algorithm:**

1. Read a and b from the user
2. If (f(a) \* f(b) > 0) then go to next step, otherwise go to step 5
3. Display invalid interval
4. Exits from the program
5. If f(a) and f(b) both gets 0, then go to next step, otherwise go to step 8
6. Display root a or b conditionally whether f(a) = 0 or not
7. Exits from the program
8. Repeat through step 9 to step 17 until error > 0.0005
9. Stores c in c\_prev
10. Store (a + b)/2 in c
11. Display i, a, b, c, f(c) in a tabloid form
12. If f(c) gets 0, display c
13. If f(c) < 0, store c in a, otherwise go to next step
14. Store c in b
15. Store c-c\_prev ‘s absolute value in error
16. If i gets 1, display “----“, otherwise go to next step
17. Display error

[End of do-while loop]

1. Display “The approximate root: c”
2. End.

**Source Code:**

#include "stdio.h"

#include "stdlib.h"

#include "math.h"

#define f(x) (pow(x, 3) - (2 \* x) - 3)

int main()

{

float a = 0, b = 0, c, c\_prev, error = 0;

int i = 0;

printf("-----------------\n");

printf("Bisection Method\n");

printf("-----------------\n");

printf("Enter the Intervals (2 Numbers Required): ");

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

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

{

printf("Invalid Interval Input!!!\n");

exit(1);

}

else if (f(a) == 0 || f(b) == 0)

{

printf("The root is one of the interval. Answer is: %4.6f", f(a) == 0 ? a : b);

exit(0);

}

//For Table View

printf("Iter.\ta\t\tb\t\tc\t\tf(c)\t\terror\n");

do

{

c\_prev = c;

c = (a + b) / 2; //for bisection method

printf("%2d\t%4.6f\t%4.6f\t%4.6f\t%4.6f\t", i++, a, b, c, f(c));

if (f(c) == 0)

{

printf("Root is : %4.6f\n", c);

}

else if (f(c) < 0)

{

a = c;

}

else

{

b = c;

}

//Error Calculating

error = fabs(c - c\_prev);

if (i == 1)

{

printf("----\n");

}

else

{

printf("%4.6f\n", error);

}

} while (error > 0.0005);

printf("Now, The Approximate Root is: %4.6f\n", c);

return 0;

}

**Input/Output:**

-----------------

Bisection Method

-----------------

Enter the Intervals (2 Numbers Required): 1 2

Iter. a b c f(c) error

0 1.000000 2.000000 1.500000 -2.625000 ----

1 1.500000 2.000000 1.750000 -1.140625 0.250000

2 1.750000 2.000000 1.875000 -0.158203 0.125000

3 1.875000 2.000000 1.937500 0.398193 0.062500

4 1.875000 1.937500 1.906250 0.114410 0.031250

5 1.875000 1.906250 1.890625 -0.023281 0.015625

6 1.890625 1.906250 1.898438 0.045217 0.007813

7 1.890625 1.898438 1.894531 0.010881 0.003906

8 1.890625 1.894531 1.892578 -0.006222 0.001953

9 1.892578 1.894531 1.893555 0.002324 0.000977

10 1.892578 1.893555 1.893066 -0.001950 0.000488

Now, The Approximate Root is: 1.893066

**Discussion:**

1. This program doesn’t run for a very large value.
2. The program exits when user puts a wrong interval.