

**Shahjalal University of Science and Technology, Sylhet**

**Department of Software Engineering**

***Numerical Methods Tasks***

***Task Code : NMAL-01***

Course Title : **Numerical Analysis**

Course Code : **SWE 231**

## 

Task Provider :

**Fazle Rabbi Rakib**

Lecturer,

Dept. of Software Engineering,

Institute of Information and Communication Technology,

Shahjalal University of Science and Technology, Sylhet

Prepared by :

**Al Abid Rahman**

Registration Number: **2022831019**

Dept : Software Engineering

Session:2022-23

**1.** What could be the better approach to choose the range in the bisection method?

**Answer :**

Bisection method works on an initial interval [a, b] such that f(a) \* f(b) <0. To find the initial lower and upper bounds we simply use the **Hit and Trial** method. This method works fine in general. However, finding initial bounds using this method can lead to difficulty oftentimes. For example, in the case of the quadratic function f(x) = (x+50)(x-50), roots are 50 and -50; in between the roots the function remains negative and outside the bound [-50, 50] function remains positive. So to guess initial bounds using hit and trial we have to guess around 50. So this method is not practical in this case. We need to take a better approach like the **graphical approach**. We can roughly plot a graph of the function using different function properties and see where a sign change is more likely. We can also use an **incremental search approach** where we can start with an initial guess and increase this value until we get a sign change. This approach can be implemented by writing code. If we keep the initial interval small, then the function converges to its root much faster than a larger initial interval. So we should try to keep the interval small as well.

**2.** Apply the technique to solve a problem

i)Traditional bisection method

ii)Bisection implementation(better approach)

Compare performance with the number of iterations

**(i) Traditional bisection method :**

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

**using namespace std;**

**double f(double x){**

**return (x\*x\*x - 4\*x -9) ;**

**}**

**int main() {**

**double lower=0, upper=10;**

**cout<<"Lower Bound : "<<lower<<endl<<"Upper Bound : "<<upper<<endl;**

**double tolerance = 1e-10;**

**double mid = (upper+lower)/2;**

**double root = mid;**

**int iterationCount = 0;**

**while(fabs(f(mid))>tolerance && fabs(upper-lower)>tolerance){**

**mid = (upper+lower)/2;**

**root = mid;**

**if(f(mid)\*f(lower)<0) upper = mid;**

**else lower = mid;**

**iterationCount++;**

**}**

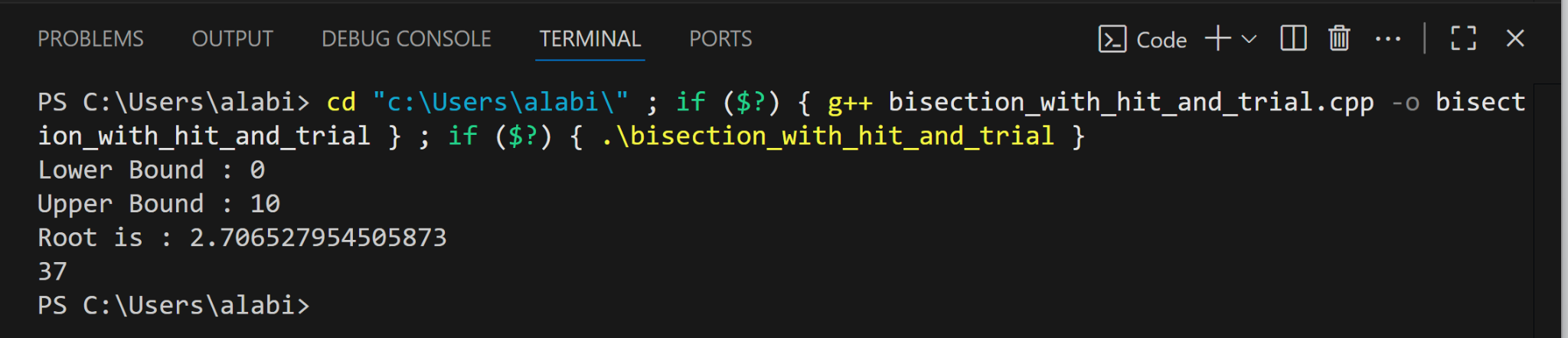
**printf("Root is : ");**

**printf("%.15f\n", root);**

**cout<<iterationCount<<endl;**

**return 0;}**

**Output :**

****

**(ii) Bisection implementation(better approach) :**

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

**using namespace std;**

**double f(double x){**

**return (x\*x\*x - 4\*x -9) ;**

**}**

**int main() {**

**double lower, upper;**

**int minInterval = INT\_MAX;**

**for(int i=-10; i<=9; i++){**

**for(int j=i+1; j<=10; j++){**

**if(f(i)\*f(j)<0) {**

**if((j-i)<=minInterval) { lower = i; upper = j; minInterval = j-i; }**

**break;**

**}**

**}**

**}**

**cout<<"Lower Bound : "<<lower<<endl<<"Upper Bound : "<<upper<<endl;**

**double tolerance = 1e-10;**

**double mid = (upper+lower)/2;**

**double root = mid;**

**int iterationCount = 0;**

**while(fabs(f(mid))>tolerance && fabs(upper-lower)>tolerance){**

**mid = (upper+lower)/2;**

**root = mid;**

**if(f(mid)\*f(lower)<0) upper = mid;**

**else lower = mid;**

**iterationCount++;**

**}**

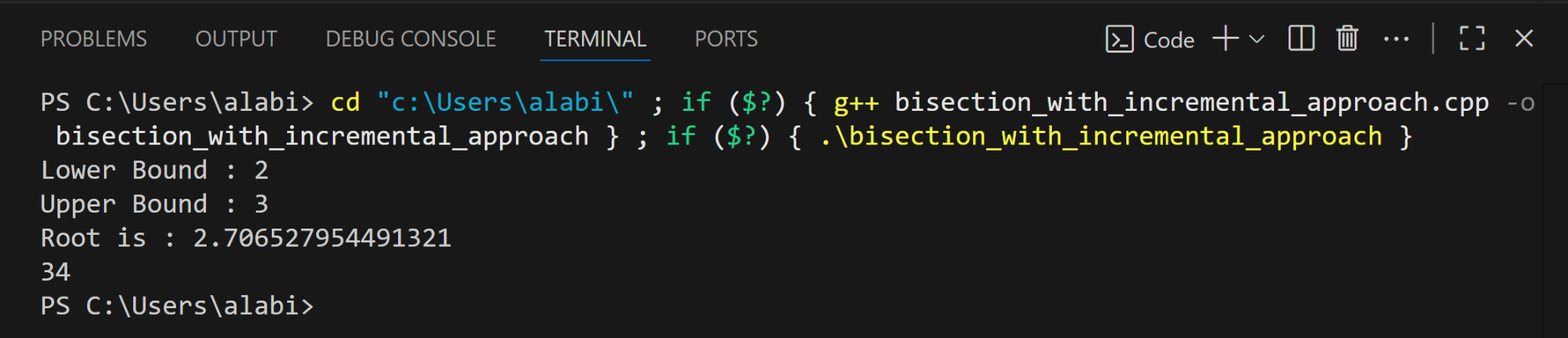
**printf("Root is : ");**

**printf("%.15f\n", root);**

**cout<<iterationCount<<endl;**

**return 0; }**

**Output :**

****

Iteration numbers in both methods are almost the same, so no practical performance gain here. This is because whatever interval we choose as initial bounds, the interval rapidly shrinks in size with each iteration as we approach the root closer and closer.