`

National University of Computer and Emerging Sciences

Department of Computer Science

**CS-325 : Numerical Computing BSCS- C Spring 2020**

**Project Report**

**Course Instructor:** Nadeem Arif Khan

**Group Members:**

* Ahmed Saqib (18K-0279)
* Hafiz Saroosh Ahmed (18K-0127)

**Programming Language:** C++

**Introduction:**

**Numerical computing** is an interconnected combination of computer science and mathematics in which we develop and analyze algorithms for solving important problems in science, engineering, medicine, and business. The purpose of this project is to implement what we studied in the semester, basically a practical approach. In our project we used C++ Programming language to implement the following numerical methods, i.e Bisection Method, Regular Falsi Method, Newton Raphson Method, Secant Method and Fixed Point Iteration.

1. **Bisection Method:**

Bisection method is a root-finding method which is applied to any continuous function for which one has to know two values with opposite signs (a<0= -ve, b>0= +ve). The method consists of repetitively bisecting the interval defined by these values and then selecting the subinterval in which the function changes sign, and therefore must contain a root.

In this method we take the mean value of **a,b** and them put that value in the function. If function gives a positive value then **b** is replace with the new value and if it gives a negative value then **a** is replaced with the new value. This process goes on until the root is approximatelyequal to *any value* in the final (very small) interval.

**Algorithm:**

Step1 : Start

Step 2: Input a, b and value of epsilon. (e= absolute error, the degree of accuracy)

Step 3: Compute the func values, func(a) and func(b).

Step 4: if (func(a)\*func(b) >=0) then print assumption of values are incorrect.

Step 5: else, compute the mean value x=(a+b)/2.

Step 6: if (func(x)\*func(a) < 0) then assign b=x, else assign a=x;

Step 7:Increment in i and loop continues with a new value.

Step 8: Stop

**C++ Program:**

#include<iostream>

#include<cmath>

using namespace std;

#define e 0.01

double func(double x)

{

return ((x\*x) - 6);

}

void bisection(double a, double b)

{

if (func(a)\*func(b) >= 0)

{

cout << "Asumption of the values are incorrect!" << endl;

return;

}

double x = a;

int i=0;

cout<< "Iteration\tValue of a\tValue of b\tValue of root"<<endl;

while (abs(b-a) >= e)

{

x = (a+b)/2;

if (func(x) == 0.0)

break;

else if (func(x)\*func(a) < 0)

b = x;

else

a = x;

cout <<i+1<<"\t\t"<<a<<"\t\t"<<b<<"\t\t"<<x<<endl;

i++;

}

cout<<endl;

cout << "Root Value: " << x;

}

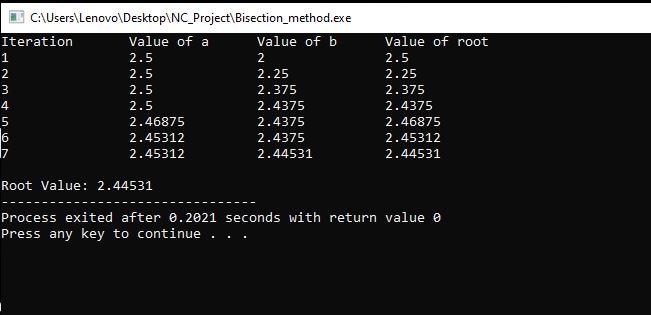
int main()

{

double a =3, b = 2;

bisection(a, b);

return 0}



1. **Regular Falsi Method**:

The Regula–Falsi Method is a numerical method for estimating the roots of a polynomial **f(x)**.   A value **x** replaces the midpoint in the Bisection Method and serves as the new approximation of a root of **f(x)**. A continuous function.

**Algorithm**:

Step 1: Start

Step 2: Input n (number of iterations)

Step 3: Declaring the values of a and b which are to be used in the function and passing it to the function.

Step 4: if (func(a) \* func(b) >= 0) then error of incorrect assumption.

Step 5: For loop for the number of iterations to be performed.

Step 6: Determine the value of c through c = (a\*func(b) - b\*func(a))/(func(b) - func(a)).

Step 7: if (func(c)\*func(a) < 0) is true then assign the value of b to a new variable c, else assign a to c.

Step 8: loop continues until the number of iterations are completed.

Step 9: Stop

**C++ Program:**

#include<iostream>

#include<math.h>

using namespace std;

int n=0;

double func(double x)

{

return ((x\*x) - 6);

}

void regulaFalsi(double a, double b)

{

if (func(a) \* func(b) >= 0)

{

cout << "You have not assumed right a and b" << endl;

return;

}

double c = a;

cout<< "Iteration\tValue of a\tValue of b\tValue of root"<<endl;

for (int i=0; i < n; i++)

{

c = (a\*func(b) - b\*func(a))/(func(b) - func(a));

if (func(c)==0)

break;

else if (func(c)\*func(a) < 0)

b = c;

else

a = c;

cout <<i+1<<"\t\t"<<a<<"\t\t"<<b<<"\t\t"<<c<<endl;

}

cout << "Root Value: " << c;

}

int main()

{

cout << "Enter number of iterations: ";

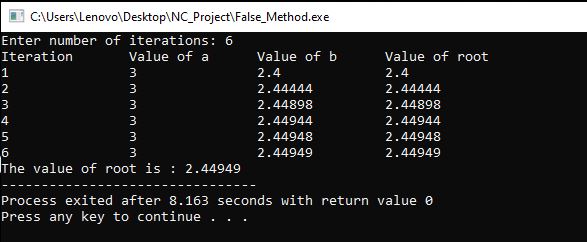
cin >> n;

double a = 3, b = 2;

regulaFalsi(a, b);

return 0;

}



1. **Secant Method:**

The secant method is a root-finding algorithm that uses a succession of roots of secant lines to better approximate a root of a function f. The secant method can be thought of as a finite-difference approximation of Newton's method.

**Algorithm:**

Step 1: Start

Step 2: Declare the values for x1 and x2 and pass through the secant function.

Step 3: if (func(x1) \* func(x2) < 0) then its true else false

Step 4: using a do while (fabs(xn - x) >= e) loop we assign the value of x = (x1 \* func(x2) - x2 \* func(x1)) / (func(x2) - func(x1));

Step 5: Assigning the product of func(x1) \* func(x) to c.

Step 6: x1 = x2; x2 = x;

Step 7: if equal to root then break

Step 8: repeat the loop

Step 9: Stop

**C++ Program:**

#include <iostream>

#include<cmath>

#define e 0.0001

using namespace std;

float func(float x)

{

return (pow(x, 3) - x - 1);

}

void secant(float x1, float x2)

{

float n = 0, xn, x, c;

cout<< "Iteration\tValue of x1\tValue of x2\tValue of root"<<endl<<endl;

if (func(x1) \* func(x2) < 0) {

do

{

x = (x1 \* func(x2) - x2 \* func(x1)) / (func(x2) - func(x1));

c = func(x1) \* func(x);

x1 = x2;

x2 = x;

cout <<n<<"\t\t"<<x1<<"\t\t"<<x2<<"\t\t"<<x<<endl;

++n;

if (c == 0)

break;

xn = (x1 \* func(x2) - x2 \* func(x1)) / (func(x2) - func(x1));

} while (fabs(xn - x) >= e);

cout<<endl;

cout << "Root Value: " << x<<endl;

cout << "No. of iterations = " << n << endl;

}

else

cout << "Can not find a root in the given interval";

}

int main()

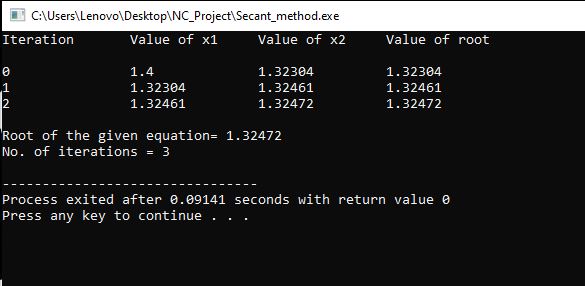
{

float x1 = 1.3, x2 = 1.4;

secant(x1, x2);

return 0;

}



4**) Newton Raphson:**

The **Newton-Raphson method** (also known as Newton's method) is a way to quickly find a good approximation for the root of a real-valued function f(x) =0.

**Algorithm:**

Step 1: Start

Step 2: Declare the value of variable x and pass through the function.

Step 3: Compute values of func(x) and derivFunc(x) for given initial x.

Step 4: Compute h: h = func(x) / derivFunc(x)

Step 5: While h is greater than e, compute h = func(x) / derivFunc(x) and x = x – h.

Step 6: Increment the value of i.

Step 7: Stop.

**C++ Program:**

#include<iostream>

#include<cmath>

#define e 0.001

using namespace std;

double func(double x)

{

return ((x\*x) - (4\*x) - 7);

}

double derivFunc(double x)

{

return 2\*x - 4;

}

void newtonRaphson(double x)

{

cout<< "Iteration\tValue of root"<<endl;

double h = func(x) / derivFunc(x);

int i=0;

while (abs(h) >= e)

{

h = func(x)/derivFunc(x);

x = x - h;

cout << i+1 << "\t\t" << x << endl;

++i;

}

cout << "Root Value: " << x;

}

int main()

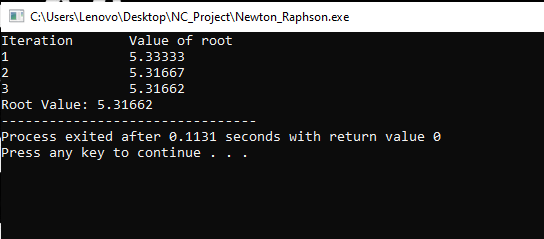
{

double x = 5;

newtonRaphson(x);

return 0;

}



5) **Fixed Point Iteration:**

In numerical analysis, **fixed**-**point iteration** is a method of computing fixed points of iterated functions.

**Algorithm:**

Step 1: Start

Step 2: Input the values and through the fixed function.

Step 3: Here the values are calculated by the formula which gives the best value.

Step 4: Value returns.

Step 5: Loop continues

Step 6: Stop

**C++ Program:**

#include<iostream>

#include<math.h>

using namespace std;

double fixed\_point(double x)

{

double k=0.25;

double j= pow(3\*x,2)+3;

double f=pow(j,k);

return f;

}

int main()

{

double x,n;

cout<<"Enter initial value between: ";

cin>>x;

cout<<"Enter Number of iteration: ";

cin>>n;

for(int i=0;i<n;i++)

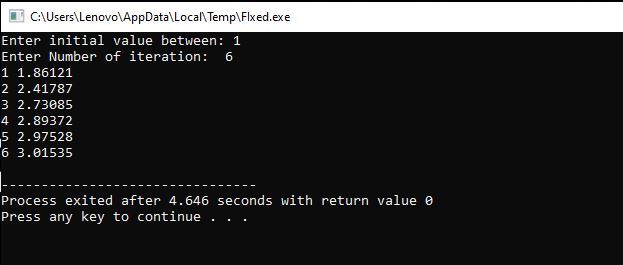
{

x=fixed\_point(x);

cout<<i+1<<" "<<x<<"\n";

}

}



**Comparison Between Regular Falsi and Secant:**

Regular Falsi iterates through intervals that always contain a root whereas the secant method is basically Newton's method without explicitly computing the derivative at each iteration. The secant is faster but may not converge at all where as regular falsi may be slow but it converges.

**Comparison Between Newton And Bisection:**

Newton's method converges much faster, but has limitations.

The bisection method is slow, but has no limitations and will always get you to the same answer eventually.

**Comparison Between Secant And Bisection:**

Secant is more faster than bisection, It converges at faster than a linear rate.