Heaven's Light is Our Guide
Rajshahi University of Engineering & Technology

Department of Computer Science & Engineering

Lab Report

Topic : Solution of Algebraic Equation

Course Title : Sessional Based on CSE 2103
Course Code : CSE 2104

Submitted By :
Md. Tanzid Hasan
Section : A
Roll No : 1603054

Submitted To :
Shyla Afroge
Assistant Professor
Dept. of Computer Science & Engineering

Date : 03-03-2018

Experiment No : 01

Experiment Name : Newton-Raphson Method

Theory :

In numerical analysis, Newton's method (also known as the Newton–Raphson method), named after Isaac Newton and Joseph Raphson, is a method for finding successively better approximations to the roots (or zeroes) of a real-valued function. It is one example of a root-finding algorithm.

$$x : f(x) = 0 .$$

The Newton–Raphson method in one variable is implemented as follows:

The method starts with a function f defined over the real numbers x, the function's derivative f', and an initial guess x_0 for a root of the function f. If the function satisfies the assumptions made in the derivation of the formula and the initial guess is close, then a better approximation x_1 is

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} .$$

Geometrically, $(x_1, 0)$ is the intersection of the x-axis and the tangent of the graph of f at $(x_0, f(x_0))$.

The process is repeated as

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

until a sufficiently accurate value is reached.

This algorithm is first in the class of Householder's methods, succeeded by Halley's method. The method can also be extended to complex functions and to systems of equations.

Code :

```
#include <iostream>

#include <math.h>


using namespace std;


double df(double e)

{
```

```
    double s=(3*e*e)-2;
    return s;
}

double f(double d)
{
    double t=(d*d*d)-(2*d)-5;
    return t;
}

int main()
{
    int i=1;
    double a,x1,x2=0,b=2.0;
    cout<<endl<<"i\t\t"<<"x\t\t"<<"Q(x)"<<endl<<endl;
    for(;;)
    {
        x1=b-(f(b)/df(b));
        cout<<i<<"\t\t"<<b<<"\t\t"<<x1<<endl;
        if(fabs(x1-x2)<0.0001)
            break;
        else
            b=x1;
        x2=x1;
        i++;
    }
    cout<<endl<<"Root : "<<x1<<endl;

}
```

Output :

i	x	Q(x)
1	2	2.1
2	2.1	2.09457
3	2.09457	2.09455

Root : 2.09455

Discussion :

In this problem, no input were taken. The equation was fixed at the beginning. After fixing the equation, two initial guess value was also assigned. Then the newton-raphson method was started as the algorithm. As the terminating condition, the difference was set 0.0001. So when the difference was less than 0.0001, the algorithm stopped and the last root was shown as result.

Experiment No : 02

Experiment Name : Ramanujan's Method

Theory :

Srinivasa Ramanujan described an iterative method which can be used to determine the smallest root of the equation like f(x) = 0 is called Ramanujan's method.

where f(x) is of the form

$$f(x) = 1- (a_1x + a_2x^2 + a_3x^3 + a_4x^4 + - - - - - - - - - -)$$

For smaller values of x, we can write

$$[1- (a_1x + a_2x^2 + a_3x^3 + a_4x^4 + - - - -)]^{-1} = b_1x + b_2x^2 + b_3x^3 + - - - - - -$$

Expanding the left-hand side by binomial theorem, we can obtain

$$1 + (a_1x + a_2x^2 + a_3x^3 + - -) + (a_1x + a_2x^2 + a_3x^3 + - -)^2 + - - = b_1x + b_2x^2 + b_3x^3 + - -$$

Comparing the coefficient of like power of x on both sides of the equation, we get

$b_1 = 1,$
$b_2 = a_1 = a_1b_1,$
$b_3 = a_1^2 + a_2 = a_1b_2 + a_2b_1,$
.
.
.
$b_n = a_1b_{n-1} + a_2b_{n-2} + - - - - + a_{n-1}b_1 \quad n = 2, 3, 4, \ldots\ldots\ldots$

Without any proof, Ramanujan states that the successive convergents, b_n/b_{n+1}, approach a root of the equation f(x) = 0, where f(x) is given.

Code :

```
#include <bits/stdc++.h>
using namespace std;
double a[1000],b[1000];
int main()
{
   double c[100];
   int n;
   cout<<"Enter the power of equation : ";
   cin>>n;
```

```
double arr[n+1];
int p=n;
for(int i=0;i<n+1;i++)
{
   cout<<"Coefficient of x^"<<p--<<" : ";
   cin>>arr[i];
}
p=n-1;
for(int i=1;i<n+1;i++)
{
   a[i]=-(arr[p]/arr[n]);
   p--;
}
b[1]=1;
int k,l,m;
for(int i=2;;i++)
{
   k=i-1;
   for( int j=1;j<i;j++)
   {
      b[i]=b[i]+(a[j]*b[k]);
      k--;
   }
   l=i;
   if(fabs(b[i]-b[i-1])<0.0000001)
      break;
}
for(int i=1;i<l;i++)
{
```

```
        c[i]=b[i]/b[i+1];
        if(i>1)
            if(fabs(c[i]-c[i-1])<0.0001)
            {
                m=i;
                break;
            }
    }
    cout<<"i\t\tb[n]\t\tb[n+1]\t\tb[n]/b[n+1]"<<endl<<"----------------------------
----------------------------"<<endl;
    for(int i=1;i<=m;i++)
        cout<<i<<"\t\t"<<b[i]<<"\t\t"<<b[i+1]<<"\t\t"<<c[i]<<endl;
    cout<<endl<<"Root : "<<c[m]<<endl;
}
```

Input :

Enter the power of equation : 3

Coefficient of x^3 : 1

Coefficient of x^2 : -6

Coefficient of x^1 : 11

Coefficient of x^0 : -6

Output :

i	b[n]	b[n+1]	b[n]/b[n+1]
1	1	1.83333	0.545455
2	1.83333	2.36111	0.776471
3	2.36111	2.66204	0.886957
4	2.66204	2.82485	0.942365
5	2.82485	2.91037	0.970616
6	2.91037	2.9545	0.985063

7	2.9545	2.97702	0.992434
8	2.97702	2.98843	0.996181
9	2.98843	2.99419	0.998077
10	2.99419	2.99709	0.999034
11	2.99709	2.99854	0.999515
12	2.99854	2.99927	0.999757
13	2.99927	2.99963	0.999878
14	2.99963	2.99982	0.999939

Root : 0.999939

Discussion :

In this problem, the equation was generated. After generating the equation, the ramanujan's method was started as the algorithm. As the terminating condition, the difference was set 0.0001. So when the difference was less than 0.0001, the algorithm stopped and the last root was shown as result.