Study on Newton Algorithm for Solving Mathematical Functions Using C-programing

Banyar Min Min Tun*[1]

Abstract

This paper presents a reasonable computer application solution of the mathematical function for Newton method in numerical methods, which have been designed and implemented of a Computerized System for Newton method. There are various numerical methods to solve equations of the type $f(x)=0$, its than we only discuss Newton - Raphson numeric method Algorithm and computerized simple program with example using C programming language.

Keywords:
Numerical methods, Newton-Raphson, C programing language

*Associate Professor, Department of ICT

1. Introduction

Numeric methods can be formulated as algorithms. An algorithm is a step-by-step procedure that states a numeric method in a form (a "pseudo code") understandable to humans. The algorithm is then used to write a program in a programming language that the computer can understand so that it can execute the numeric method. Important algorithms for Newton's method state in this paper. We give example for Newton's solving method with C programming language. For routine tasks, some other software system may contain programs (eg. Microsoft Excel) that we can use. Some of Numeric Methods could not use readymade software, so we should create a solving program. In our studying paper, to create a C programming language for Newton's numerical method, start basic from Newton-Respon's Method solving equation $f(x)=0$.

1.2 Objectives of the study

The objectives of the study are-

(i) To Develop Algorithm for Solving Mathematical Equations

(ii) In Newton's Method, To Solve Mathematical functions Using C-programming language.

1.3. Method of Study

This paper mainly use in literature survey and the data sources are based on secondary data obtaining from libraries, internet, website. We develop the Newton's algorithm for numerical data and that data solved in visual basic compiler using C-programing language.

1.4. Scope of the study

This study is focused on numerical method can compute easy computerize system.

2. Solution of single Equations

In this paper we select basic kinds of problems and discuss numeric methods on how to solve them. We will learn about a variety of important problems and become familiar with ways of thinking in numerical analysis. We choose $\boldsymbol{f(x)=0}$ problem because this problem is perhaps the easiest conceptual problem to find solutions of a single equation.

$\boldsymbol{f(x)=0}$ **,** Where f is a given function.

A solution of $\boldsymbol{f(x)=0}$ is a number x=s such that $\boldsymbol{f(s) = 0}$ Here, **s** suggests "solution," but we shall also use other letters. eg. $\boldsymbol{f(a) = 0,\ f(b) = 0,\ \ldots..}$

It is interesting to note that the task of solving $f(x)=0$ is a question made for numeric algorithms, as in general there are no direct formulas, except in a few simple cases.

2.1 Examples of single equations

Some examples of single equations are $x^3 + x = 1$, $\sin x = 0.5x$, $tanx = x$, $\cosh x = \sec x$, which can all be written in the form of $f(x)=0$. The first of the four equations is an algebraic equation because the corresponding f is a polynomial. In this case the solutions are called roots of the equation and the solution process is called finding roots. The other equations are transcendental equations because they involve transcendental functions. In this case the solutions are called roots of the equation and the solution process is called finding roots.

2.2 Solution of *f*(x)=0 by Iteration Method

To solve $f(x)=0$, when there is no formula for the exact solution available. $f(x)=0$ have many solution so we can use an approximation method, such as an **iteration method**. Iteration method is a method in which we start from an initial guess x_0 (which may be poor) and compute step by step x_0, x_1, x_2, x_3,…..(in general better and better) approximations of an unknown solution of $f(x)=0$.

In general, iteration methods are easy to program because the computational operations are the same in each step—just the data change from step to step—and, more importantly, if in a concrete case a method converges, it is stable. We discuss **Newton's Methods** in this paper among from other many iteration methods.

3. Newton's Method for Solving Equation f(x)=0

Newton's method, also known as Newton–Raphson's method, is iteration method for solving equations f(x)=0 where f is assumed to have a continuous derivative f'. The method is commonly used because of its simplicity and great speed.

The idea is that we approximate the graph of f, by suitable tangents. Using an approximate value obtained from the graph of f, we let be the point of intersection of the x-axis and the tangent to the curve of f at x_0 (see Fig.1.1). Then

$$\tan\beta = f'(x_0) = \frac{f(x_0)}{x_0 - x_1}, \qquad \text{hence} \qquad x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}.$$

To implement it analytically we need a formula for each approximation in terms of the previous one, i.e. we need x_{n+1} in terms of x_n.

In fig (1.1) ,

The equation of the tangent line to the graph y = $f(x)$ at the point (x_0, $f(x_0)$) is

$$y - f(x_0) = f'(x_0)(x - x_0)$$

The tangent line intersects the x-axis when y = 0 and x = x_1, so

$$-f(x_0) = f_0\,(x_0)(x_1 - x_0)$$

Solving this for x_1 gives

$$x_1 = x_0 - \frac{f(x_0)}{f\prime\,(x_0)}\,,$$

`

$$x_2 = x_1 - \frac{f(x_1)}{f\prime\,(x_1)}$$ and,

more generally,

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

The above formula application is to solving equations of the form $f(x) = 0$. It is called the Newton Raphson method.

It is guaranteed to converge if the initial guess x_0 is close enough, but it is hard to make a clear statement about what we mean by 'close enough' because this is highly problem specific. A sketch of the graph of $f(x)$ can help us decide on an appropriate initial guess x_0 for a particular problem.

Then we can be obtained Taylor's formula from algebraically solving above equation.

$$f(x_{n+1}) \approx f(x_n) + (x_{n+1} - x_n)f'(x_n) = 0$$

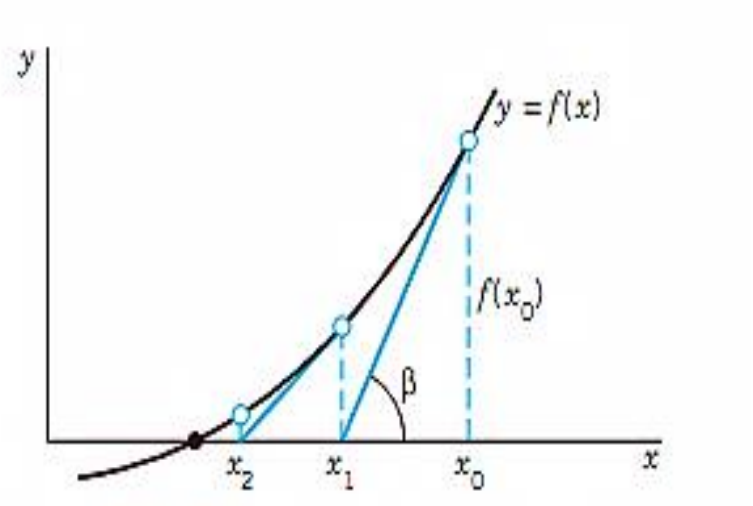

Fig: 1.1 Newton's Method

4. Newton's Algorithm for solving Equations f(x) =0

An algorithm is a finite set of precise instruction for performing a computation or for solving problem.

There are(5) steps in this algorithm. Newton-Raphson's method has mainly potions for computing is derivative of $f(x)$. Derivative of $f(x)$ must be consist (ie. $f'(x_0)\neq 0$).

If $f'(x_0)=0$, the computing procedure will be stop and unsuccessfully. Step (2) is main step for this algorithm (Table 1.1)

Step (5) in this algorithm can also be obtained if we algebraically solve Taylor's formula

Table (1.1) Newton's method for solving Equations $f(x) =0$

ALGORITHM NEWTON $(f, f', x_0, \epsilon, N)$

This algorithm computes a solution of $f(x) = 0$ given an initial approximation x_0 (starting value of the iteration). Here the function $f(x)$ is continuous and has a continuous derivative $f'(x)$.

INPUT: f, f', initial approximation x_0, tolerance $\epsilon > 0$, maximum number of iterations N.

OUTPUT: Approximate solution x_n ($n \leqq N$) or message of failure.

For $n = 0, 1, 2, \cdots, N - 1$ do:

1 Compute $f'(x_n)$.

2 If $f'(x_n) = 0$ then OUTPUT "Failure." Stop.

[*Procedure completed unsuccessfully*]

3 Else compute

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

4 If $|x_{n+1} - x_n| \leqq \epsilon |x_{n+1}|$ then OUTPUT x_{n+1}. Stop.

[*Procedure completed successfully*]

End

5 OUTPUT "Failure". Stop.

[*Procedure completed unsuccessfully after N iterations*]

End NEWTON

5. Example of Newton's method to solve the equation f(x) = 0, where $f(x)=\frac{1}{x}-2^x$

In the following calculation, the newton's methods discussed are used to solve the equation f(x) = 0, where $\boldsymbol{f(x)=\frac{1}{x}-2^x}$

Let us solve $\frac{1}{x}-2^x=0$ for x.

In this case $f(x)=\frac{1}{x}-2^x$, $f'(x)=-\frac{1}{x^2}-(2^x \ln 2)$

Substitute in Newton's method

$$x_{n+1}=x_n-\frac{\left(\frac{1}{x}-2^x\right)}{\left(-\frac{1}{x^2}-(2^x \ln 2)\right)}$$

$$x_1=x_0-\frac{\left(\frac{1}{x_0}-2^{x_0}\right)}{\left(-\frac{1}{{x_0}^2}-(2^{x_0} \ln 2)\right)}$$

We can get $f(x)$ solutions choosing the value of x. There are many solutions in this problem set we compute carrying on start form x_1, x_2 etc. We can stop when the digits stop changing to the required degree of accuracy.

6. Example of C- programming for solve the equation f(x) = 0

In the following calculation, the newton's methods discussed are used to solve the equation f(x) = 0, where $\boldsymbol{f(x)=\frac{1}{x}-2^x}$

We discuss these programs in a Linux environment (their discussion in any Unix environment would be virtually identical). The following file funct .c will contain the definition of the function f:

```
1 #include <math.h>
2
3 double funct(double x)
4 {
5 double value;
6 value = 1.0/x-pow(2.0,x);
7 return(value);
8 }
```

Here pow(x,y) is C's way of writing xy. Using decimal points (as in writing 1.0 instead of 1), we indicate that the number is a floating point constant. The numbers at the beginning of these lines are not part of the file; they are line numbers that are occasionally useful in a line-by-line discussion of such a file.

The program for bisection is in the file bisect.c, to be discussed later. The simplest way of compiling these programs is to write a make file. In the present case, the make file is called (surprise) make file (this is the default name for a make file), with the following content:

```
1 all: bisect
2 bisect : funct.o bisect.o
3 gcc -o bisect -s -O4 funct.o bisect.o -lm
4 funct.o : funct.c
5 gcc -c -O4 funct.c
6 bisect.o : bisect.c
7 gcc -c -O4 bisect.c
```

Line 1 here describes the file to make; namely, the file bisect. This file contains the compiled program, and the program can be run by typing its name on the command line, that is, by typing $ bisect.

Here the dollar sign $ is the customary way of denoting the command line prompt of the computer, even though in actual life the command line prompt is usually different.

The second line contains the dependencies; the file bisect to be made depends on the files funct.o and bisect.o (the .o suffix indicates that these are object files, that is, already compiled programs – read on).

The file on the left of the colon is called the target file, and the files on the right are called the source files.

When running the make command, by typing, say,$ make all on the command line, the target file is created only if it does not already exist, or if it predates at least one of the source files (i.e., if at least one of the source files has been change since the target file has last been created). Clearly, if the source files have not changed since the last creation of the target file, there is no need to create the target file again.

Line 3 contains the rule used to create the target. One important point, a quirk of the make command, that the first character of line three is a tab character (which on the screen looks like eight spaces); the rule always must start with a tab character. The command on this line invokes the gcc compiler (the GNU C compiler) to link the already created programs funct.o and bisect.o, and the mathematics library (described as -lm at the end of the line). The -o option gives the name b isect to the file produced by the compiler. The option -s gives is passed to the loader to strip all symbols from the compiled program, thereby making the compiled program more compact. The option -O4 (the first character is “Oh” and not “zero”) specifies the optimization level.

Lines 4 and 6 contain the dependencies for creating the object files funct.o and bisect.o, and lines 5 and 7 describe the commands issued to the GNU C compiler to create these files from the source files funct.c and bisect.c. The compiler option –c means compile but do not link the assembled source files. These latter two files need to be written by the programmer; in fact, the file funct.c has already been described, and we will discuss the file bisect.c next:

7. C- programming for Newton's method to solve the equation $f(x)= \frac{1}{x} - 2^x$ with f(x) = 0

For Newton's method, in addition to the function, its derivative also needs to be calculated. The program to do this constitutes the file dfunct.c:

```
1 #include <math.h>
2
3 double dfunct(double x)
4 {
5 double value;
6 value = -1.0/(x*x)-pow(2.0,x)*log(2.0);
7 return(value);
8 }
```

The program itself is contained in the file newton.c. The makefile to compile this program is as follows:

```
1 all: newton
2 newton : funct.o newton.o dfunct.o
3 gcc -o newton -s -O4 funct.o dfunct.o newton.o -lm
4 funct.o : funct.c
5 gcc -c -O4 funct.c
6 dfunct.o : dfunct.c
7 gcc -c -O4 dfunct.c
8 gcc -c -O4 bisect.c
9 newton.o : newton.c
10 gcc -c -O4 newton.c
```

The file newton.c itself is as follows:

```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define absval(x) ((x) >= 0.0 ? (x) : (-(x)))
5
6 double funct(double);
7 double dfunct(double);
8
9 double newton(double (*fnct)(double), double (*deriv)(d
10 double startingval, double xtol, int maxits, double
11 int *itcount, int *outcome);
12
13 main()
14 {
15 /* This program implements the Newton's method
16 for solving an equation funct(x)=0. The function
```

```
17 funct() and its derivative dfunct() is defined
18 separately. */
19 const double tol=5e-10;
20 double x, fx, root;
21 int its, success;
22 root = newton(&funct, &dfunct,
23 3.0, tol, 50, &fx,
24 &its, &success);
25 if ( success == 2 ) {
26 printf("The root is %.12f. The value of "
27 " the function\nat the root is %.12f.\n", root, fx);
28 printf("%u iterations were used to find "
29 " the root\n", its);
30 }
31 else if (success == 1) {
32 printf("The derivative is too flat at %.12f\n", x);
33 }
34 else if (success == 0) {
35 printf("The maximum number of iterations has been reached\n");
36         }
37 }
38
39 double newton(double (*fnct)(double), double (*deriv)(double),
40 double startingval, double xtol, int maxits, double *fx,
41 int *itcount, int *outcome)
42 {
43 double x, dx, dfx, assumedzero=1e-20;
44 *outcome = 0;
45 x = startingval;
46 for (*itcount = 0; *itcount < maxits; (*itcount)++) {
47 dfx = deriv(x);
48 if ( absval(deriv(x))<=assumedzero ) {
49 *outcome = 1; /* too flat */
50 break;
51 }
52 *fx = fnct(x);
53 /* The next two lines are included so as to monitor
54 the progress of the calculation. These lines
55 should be deleted from a program of
56 practical utility. */
57 printf("Iteration number %2u: ", *itcount);
58 printf("x=% .12f and f(x)=% .12f\n", x, *fx);
```

```
59 dx = -*fx/dfx; x = x+dx;
60 if ( absval(dx)/(absval(x)+assumedzero) <= xtol ) {
61 *outcome = 2; /* within tolerance */
62 *fx = fnct(x);
63 break;
64       }
65 }
66 return x; /* returning the value of the root */
67 }
```

In lines 22-24 calls the bisection function, located in lines 39–67, with initial approximation x1 = 3.

The variable success keeps track of the outcome of the calculation, the value 2 indicates that the root has been successfully calculated within the required tolerance, given as $5 \cdot 10{-}10$ by the constant tol specified in line 19.

The value 1 indicates that the tangent line is too flat at the current approximation (in which case the next approximation cannot be calculated with sufficient precision).

Finally, the value 0 indicates that the maximum number of iterations (50) at present, specified in the calling statement in line 23) has been exceeded.

This program produces the following output:

```
1 Iteration number 0: x= 3.000000000000 and f(x)=-7.666666666667
2 Iteration number 1: x= 1.644576458341 and f(x)=-2.518501258529
3 Iteration number 2: x= 0.651829979669 and f(x)=-0.037017477051
4 Iteration number 3: x= 0.641077330609 and f(x)= 0.000380944220
5 Iteration number 4: x= 0.641185733066 and f(x)= 0.000000040191
6 Iteration number 5: x= 0.641185744505 and f(x)= 0.000000000000
7 The root 0.641185744505. The value of the function
8 at the root is -0.000000000000.
9 5 iterations were used to find the root
```

Conclusion

In this paper, we developed the Newton Algorithm and solved the mathematical functions in numerical methods with examples, when solving problems set, we occurred errors in the result. There are generally,

(i) Approximations in final results of computations of unknown quantities; that is, they are not exact but involve errors or round off errors.

(ii) Experimental errors are errors of given data (probably arising from measurements).

(iii) Truncating errors result from truncating (prematurely breaking off), for instance, if we replace a Taylor series with the sum of its first few terms. These errors depend on the computational method used and must be dealt with individually for each method.

(iv) An algorithm instability. However, if small changes in the initial data can produce large changes in the final results, we call the algorithm unstable.

To reduce errors we will notices the following conditions;

(i) To solve experimental errors we can use exactly data.

(ii) Stability. To be useful, an algorithm should be stable; that is, small changes in the initial data should cause only small changes in the final results.

Acknowledgements

First of all, I would like to express my deeply gratitude to Dr. Ye Ye Win (Rector) Co-operative University, Thanlyin , U Oo Tin Thein(pro rector) Co-operative University and Daw Myint Myint Sein Thanlyin(pro rector) Co-operative University, for their kind permission to conduct this research work. Special thanks are also due to Dr.Kyaw May Oo, chairperson of paper reading seminar, (professor) head of faculty of Computing , University of Information Technology, Yangon, and Dr. Ohn Mar san (professor) head of ICT Department, Co-operative University, Thanlyin who gives understanding and encouragements.

References

1. ATTILA MATE, Introduction to Numerical Analysis with C programs, August 2014.

2. Kreyszing, E, Advanced Engineering Mathematics, John Wiley, 10th Edition.

3. Kennclh.Rosen. Discrete Mathematics and its Application, 4th Edition