COSC 528 Design and Analysis of Algorithms

Project Title: Newton's Method

Assignment Due: May 11th, 2021

Faculty: Dr. James Gil De Lamadrid

Team Members:

Syed Ali

Olabode Ajishe

Contents

Introduction 2
Algorithm 3
Analysis 3
Test 4
Appendix 6
References 8

Introduction

Newton's method is a root-finding algorithm which is extensively used in the field of numerical methods. The root-finding algorithm produces better approximations to the roots of a real valued function. The basic version begins with a single-variable function *f* defined for a real variable x, the function's derivative *f'*, and an initial guess $x_0$for a root of *f*. If the function gratifies adequate assumptions and the initial guess is close, then:

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

would be considered a better approximation of the root than x_0. If we think about this geometrically $(x_1, 0)$ is considered to be intersection point of the x-axis and the tangent of the graph of *f* at (x_0, *f*(x_0)), which is that the improved guess is the unique root of the linear approximation at the initial point. The process is repeated as:

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

until a sufficiently precise value is reached based on the error value. The derivation of Newton's method is provided below:

From Taylor's theorem we know that any function $f(x)$ which has a continuous second derivative can be represented by an expansion about a point that is close to a root of $f(x)$. Suppose if we consider this root as α. Then the expansion of $f(\alpha)$ about x_n is follows

$$f(\alpha) = f(x_n) + f'(x_n)(\alpha - x_n) + R_1 \text{ – (1)}$$

Where the Lagrange form of the Taylor series expansion reminder is:

$$R_1 = \frac{1}{2!} f"(\zeta_n)(\alpha - x_n)^2$$

Where ζ_n is in between x_n and α

Since α is the root, equation (1) becomes:

$$0 = f(\alpha) = f(x_n) + f'(x_n)(\alpha - x_n) + \tfrac{1}{2} f''(\zeta_n)(\alpha - x_n)^2 \text{ – (2)}$$

Dividing equation (2) by $f'(x_n)$ and upon rearranging gives as follows:

$$\frac{f(x_n)}{f'(x_n)} + (\alpha - x_n) = -\frac{f''(\zeta_n)}{2f'(x_n)}(\alpha - x_n)^2 \text{ – (3)}$$

Newton's method is derived assuming that since $(\alpha - x_n)$ then $(\alpha - x_n)^2$ is going to be much smaller therefore:

$$\frac{f(x_n)}{f'(x_n)} + (\alpha - x_n) = 0$$

Upon further rearranging gives:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} - (4)$$

Algorithm

The algorithm for Newton's method is given below:

Let $f: \mathbb{R} \to \mathbb{R}$ be a differentiable function. The following algorithm computes an approximate solution x^* to the equation $f(x) = 0$.

Choose an initial guess x_0:

for $n = 0, 1, 2, \ldots.\, do$

if $f(x_n)$ is adequately small then

$x^* = x_n$

return x^*

end

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

if $|x_{n+1} - x_n|$ or $(|x_{n+1} - x_n| > \mathcal{E}\ and\ i < N)$ is sufficiently small, then

$x^* = x_{n+1}$

return x^*

end

end

Analysis

I: Convergence Analysis

Newton's method exhibits a quadratic convergence, and most theorists agree with it as well, below is the proof:

So, starting from Equation 3:

$$\frac{f(x_n)}{f'(x_n)} + (\alpha - x_n) = -\frac{f''(\zeta_n)}{2f'(x_n)}(\alpha - x_n)^2 - (3)$$

we can find that:

$$\underbrace{(\alpha - x_n)}_{\varepsilon_{n+1}} = -\frac{f''(\zeta_n)}{2f'(x_n)}\underbrace{(\alpha - x_n)^2}_{\varepsilon_n}$$

That is:

$$|\varepsilon_{n+1}| = \frac{-f''(\zeta_n)}{2|f'(x_n)|} * \varepsilon_n{}^2 - (5)$$

Upon taking absolute values at both sides gives the following:

$$|\varepsilon_{n+1}| = \frac{|f''(\zeta_n)|}{2|f'(x_n)|} * \varepsilon_n{}^2 - (6)$$

Thus, from equation 6 we can see that the rate of convergence is indeed quadratic if it meets the following conditions:

1) $f'(x) \neq 0$ for all $x \in I$, where I is the interval from $[\alpha - r, \alpha + r]$ for some $r \geq |\alpha - x_0|$
2) $f''(x)$ is continuous for all $x \epsilon I$
3) x_0 is **sufficiently close** to the root α

Where sufficiently close has the following meaning:

a) Taylor approximations is accurate enough such that we can ignore all the higher terms.
b) $\frac{1}{2}\left|\frac{f''(x_n)}{f'(x_n)}\right| < C\left|\frac{f''(\alpha)}{f'(\alpha)}\right|$, for some C < ∞.
c) $C\left|\frac{f''(\alpha)}{f'(\alpha)}\right| \varepsilon_n < 1$, for n ∈ ℤ, n ≥ 0 and C satisfying condition b.

II: Time Complexity

According to theoretical computer scientists, the average time complexity for Newton's method is $O(log_2N)$, however, some theorists argue the following:

1) $O(log_2N)$ is just a highest order term. The actual number of steps to convergence might be as follows:

   Constant term + coefficient * log_2N

2) There is also talk about the exact floating-point computations where the convergence might occur a step sooner or later, before the x and y converge to within the "machine epsilon" (i.e., so close that it cannot be distinguished).

Test

We were asked to approximate the following values using Newton's method and they are:

1) $\sqrt{2}$
2) sin (0.75)

rearranging problems 1 and 2 will give us:

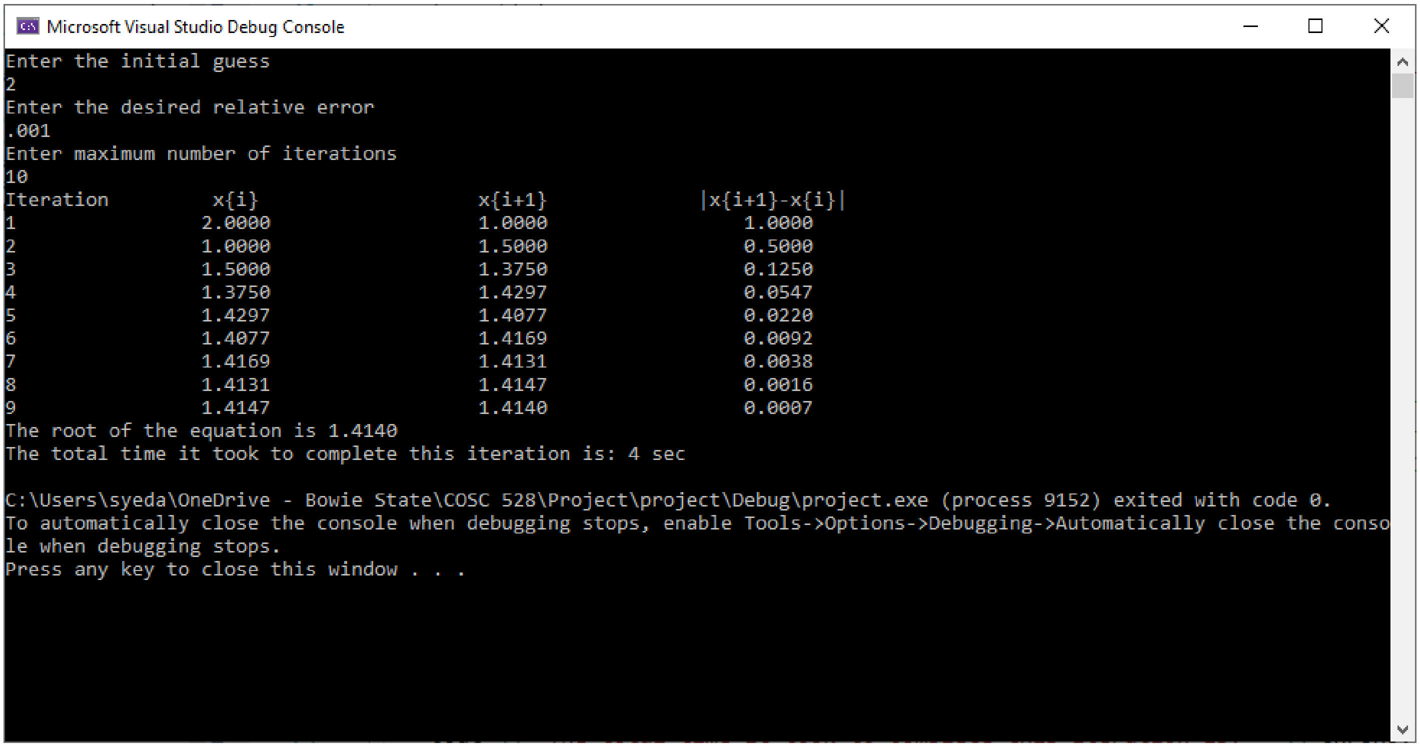
1) $f(x) = x^2 - 2$
2) $f(x) = \sin(x) - 0.68163876$

We were given a relative error of 0.001.

Implementing the method in C++ gave us the following results:

1) $f(x) = x^2 - 2$

With an initial guess of 2, after about 10 iterations we got an approximate value of 1.4140. The total time it took to complete 10 iterations is 4 seconds. Below is the output after running C++ program:



```
Microsoft Visual Studio Debug Console
Enter the initial guess
2
Enter the desired relative error
.001
Enter maximum number of iterations
10
Iteration        x{i}                x{i+1}             |x{i+1}-x{i}|
1                2.0000              1.0000               1.0000
2                1.0000              1.5000               0.5000
3                1.5000              1.3750               0.1250
4                1.3750              1.4297               0.0547
5                1.4297              1.4077               0.0220
6                1.4077              1.4169               0.0092
7                1.4169              1.4131               0.0038
8                1.4131              1.4147               0.0016
9                1.4147              1.4140               0.0007
The root of the equation is 1.4140
The total time it took to complete this iteration is: 4 sec

C:\Users\syeda\OneDrive - Bowie State\COSC 528\Project\project\Debug\project.exe (process 9152) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

Figure 1: Newton's method applied to Problem 1.

2) $f(x) = \sin(x) - 0.68163876$

With an initial guess of 1, after about 3 iterations we got an approximate value of 0.75. The total time it took to complete 3 iterations is 10 seconds. Below is the output after running C++ program:

```
Microsoft Visual Studio Debug Console
Enter the initial guess
1
Enter the desired relative error
.001
Enter maximum number of iterations
10
Iteration        x{i}                x{i+1}           |x{i+1}-x{i}|
1                1.0000              0.7042               0.2958
2                0.7042              0.7491               0.0449
3                0.7491              0.7500               0.0009
The root of the equation is 0.7500
The total time it took to complete this iteration is: 10 sec

C:\Users\syeda\OneDrive - Bowie State\COSC 528\Project\project\Debug\project.exe (process 18552) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

:

Figure 2: Newton's method applied to Problem 2.

If we compare problems 1 and 2, we notice that it takes roughly twice the amount of time for the solution to converge in problem 2 than in problem 1, we also notice that the solution tends to converge in a quadratic form thus proving the hypothesis that Newton's method exhibits a quadratic convergence.

Appendix

Problem 1

```
#include<iostream>
#include<cmath>
#include<iomanip>
#include<string>
#include<chrono>
using namespace std;
double f(double x);  // Declare the function for the given equation
double f(double x); // Define the function here, i.e. give the equation
{
	double a = sin(x) - 0.68163876; // Write the first derivative of the equation
	return a;
}
double fprime(double x);
double fprime(double x)
{
	double b = cos(x);   // Write the first derivative of the equation
	return b;
}
int main()
{
```

```
        double x{}, x1, e, fx, fx1;
        int step = 0, N;
        auto start = chrono::steady_clock::now();
        cout.precision(4);
        // Set the precision
        cout.setf(ios::fixed);
        cout << "Enter the initial guess\n"; // Take an initial guess
        cin >> x1;
        cout << "Enter the desired relative error\n";// Take the desired accuracy
        cin >> e;
        cout << "Enter maximum number of iterations\n";
        cin >> N;
        fx = f(x);
        fx1 = fprime(x);
        cout << "Iteration" << "         " << "x{i}" << "                   " << "x{i+1}"
<< "              " << "|x{i+1}-x{i}|" << endl;
do
        {
                x = x1;// make x equal to the last calculated value of x1
                step = step + 1;
                if (step > N)
                {
                        cout << "Not Convergent.";
                        exit(0);
                }
                fx = f(x);    // simplifying f(x) to fx
                fx1 = fprime(x); // simplifying fprime(x) to fx1
                x1 = x - (fx / fx1); // calculate x{1} from x, fx and fx1
                cout << step << "                 " << x << "                   " << x1
                        << "                   " << abs(x1 - x) << endl;
        } while (fabs(x1 - x) >= e); // if |x{i+1} - x{i}| remains greater than the
desired accuracy, continue the loop
        cout << "The root of the equation is " << x1 << endl;
        auto end = chrono::steady_clock::now();
        cout << "The total time it took to complete this iteration is: " <<
chrono::duration_cast<chrono::seconds>(end - start).count() << " sec\n";
return 0;
}
```

Problem 2

```
#include<iostream>
#include<cmath>
#include<iomanip>
#include<string>
#include<chrono>
using namespace std;
double f(double x);  // Declare the function for the given equation
double f(double x); // Define the function here, i.e. give the equation
{
        double a = pow(x, 2.0) - 2.0; // Write the first derivative of the equation
        return a;
}
double fprime(double x);
double fprime(double x)
{
        double b = 2 * (x, 1.0);    // Write the first derivative of the equation
        return b;
```

```
}

int main()
{
	double x{}, x1, e, fx, fx1;
	int step = 0, N;
	auto start = chrono::steady_clock::now();
	cout.precision(4); // Set the precision
	cout.setf(ios::fixed);
	cout << "Enter the initial guess\n"; // Take an initial guess
	cin >> x1;
	cout << "Enter the desired relative error\n";// Take the desired accuracy
	cin >> e;
	cout << "Enter maximum number of iterations\n";
	cin >> N;
	fx = f(x);
	fx1 = fprime(x);
	cout << "Iteration" << "          " << "x{i}" << "                    " << "x{i+1}"
<< "              " << "|x{i+1}-x{i}|" << endl;
do
	{
		x = x1;// make x equal to the last calculated value of x1
		step = step + 1;
		if (step > N)
		{
			cout << "Not Convergent.";
			exit(0);
		}
		fx = f(x);    // simplifying f(x) to fx
		fx1 = fprime(x); // simplifying fprime(x) to fx1
		x1 = x - (fx / fx1); // calculate x{1} from x, fx and fx1
		cout << step << "                     " << x << "                   " << x1
			<< "                   " << abs(x1 - x) << endl;
	} while (fabs(x1 - x) >= e); // if |x{i+1} - x{i}| remains greater than the
desired accuracy, continue the loop
	cout << "The root of the equation is " << x1 << endl;
	auto end = chrono::steady_clock::now();
	cout << "The total time it took to complete this iteration is: " <<
chrono::duration_cast<chrono::seconds>(end - start).count() << " sec\n";
return 0;
}
```

References

- https://www.math.usm.edu/lambers/mat460/fall09/lecture10.pdf
- https://en.wikipedia.org/wiki/Newton%27s_method
- https://courses.csail.mit.edu/6.006/fall11/rec/rec12_newton.pdf
- https://en.citizendium.org/wiki/Newton's_method#Computational_complexity