Yuchen Jin

CUS1188

Newton's Iteration

Introduction:

Roots of functions are input values that result in the function evaluating to zero. Graphically, roots can be located where the graph of a function intersects with the x-axis. How can calculators and computers precisely calculate roots of functions? Newton's Iteration is one possible way. Also known as Newton's Method and the Newton-Raphson Method, it is a numerical analysis method of closely approximating real roots of smooth or differentiable functions. This can be applied to calculate roots of numbers and it can also be applied to precisely approximate division and reciprocals (Weisstein).

Concept:

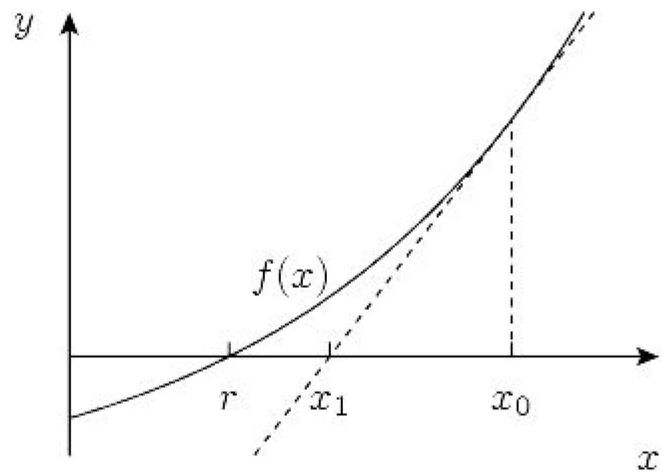
Newton's Iteration estimates roots of functions using its derivative. A derivative represents the slope of the tangent line at a particular point along the function. It begins with an initial given estimate of the root, x_0. Find the point at which the line tangent to the function at x_0 intersects with the x-axis. This becomes the new estimate, x_1 for the next iteration. This process is repeated until the estimate converges (Walls).

Recurrence Equation:

Below is the equation that describes the Newton Iteration algorithm. x_i represents the estimate for the root at each iteration.

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

This equation can be derived as follows:



(source)

The slope of the tangent line at ($x_0, f(x_0)$) , $f'(x_0)$, can be expressed using the following equation: $f'(x_0) = \frac{f(x_0) - 0}{x_0 - x_1}$, obtained by inserting the points ($x_0, f(x_0)$) and ($x_1, 0$) into the slope formula $m = \frac{y_2 - y_1}{x_2 - x_1}$.

Solve the slope equation for x_1 :

$$(x_0 - x_1) f'(x_0) = f(x_0) - 0$$

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

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

If we derive the equation for x_2 , we will end up with $x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}$ and so on for $x_3, x_4 \ldots$. The sequence can then be summarized using the Newton Iteration equation (Weisstein).

Division and Reciprocals:

One of the most common applications of this algorithm is calculating divisions and reciprocals. Divisions can be expressed as c/a or $a \times \frac{1}{a}$ (and a reciprocal of an integer a is $1/a$). The key to division and reciprocals is calculating the $\frac{1}{a}$, thus we will express this as $\frac{1}{a} = x$.

$$a - \frac{1}{x} = 0$$

$$f(x) = a - x^{-1}$$

$$f'(x) = x^{-2}$$

Substitute *f(x)* and *f'(x)* into the recurrence equation and simplify:

$$x_{n+1} = x_n - \frac{a - x_n^{-1}}{x_n^{-2}}$$

$$x_{n+1} = x_n(2 - ax_n)$$

Example 1:

Use Newton's Iteration to estimate $\sqrt[5]{1000}$. Start with $x_0 = 3.5$ and estimate the root up to x_4.

$$\sqrt[5]{1000} = 1000^{\frac{1}{5}} = x$$

$$x - 1000^{\frac{1}{5}} = 0$$

$$f(x) = x^5 - 1000$$

$$f'(x) = 5x^4$$

$$x_1 = 3.5 - \frac{f(3.5)}{f'(3.5)} = 4.132778009$$

$$x_2 = 4.132778009 - \frac{f(4.132778009)}{f'(4.132778009)} = 3.991807286$$

$$x_3 = 3.991807286 - \frac{f(3.991807286)}{f'(3.991807286)} = 3.9811209306$$

$$x_4 = 3.9811209306 - \frac{f(3.9811209306)}{f'(3.9811209306)} = 3.981071707$$

Root calculated on a TI-84 calculator = 3.981071706

Example 2

Use Newton's Iteration to estimate $\frac{1}{115}$. Start with $x_0 = .001$ and estimate to x_5

$x_1 = .001(2 - 115 \cdot .001) \ = .001885$

$x_2 = .001885(2 - 115 \cdot .001885) = .0033613791$

$x_3 = .0033613791(2 - 115 \cdot .0033613791) = .0054233882$

$x_4 = .0054233882(2 - 115 \cdot .0054233882) = .0074642653$

$x_5 = .0074642653(2 - 115 \cdot .0074642653) \ = .0085212761$

Quotient calculated on a T1-84 calculator = .0086956522

Quadratic Convergence

The rate of convergence represents how quickly a converging sequence approaches its limit. Newton's Iteration's rate of convergence is quadratic (Weisstein). Quadratic convergence means that generally, Newton's Iteration will converge faster compared to other rates of convergence, such as linear or superlinear. It also means that the number of correct digits in an estimate roughly doubles with each iteration of the algorithm (Peng).

The Newton Iteration convergence rate can be expressed as:

$$\tfrac{1}{a} - x_{n+1} = a(\tfrac{1}{a} - x_n)^{\,2} \qquad \text{(Salvy).}$$

Code Implementation:

```
# basic implementation of newton iteration algorithm
def newton(func, estimate, min_error=.0001):
    diff = func(estimate)/derivative(func, estimate)
    while abs(diff) > min_error:
        diff = func(estimate)/derivative(func, estimate)
        estimate = estimate - diff
        diff = func(estimate)/derivative(func, estimate)
        max_iterations -= 1
    return estimate
```

```
# newton iteration for division
def newton_div(a, min_error=.0001):
    k = 0
    while 10**k < a:
        k += 1
    init_estimate = 1/(10**k)
    estimate = init_estimate * (2 - (a * init_estimate))
    diff = estimate - init_estimate
    while min_error < abs(diff):
        old_estimate = estimate
        estimate = estimate * (2 - (a * estimate))
        diff = estimate - old_estimate
    return estimate
```

Time Complexity:

At first glance, one might think that the time complexity of the basic implementation of Newton's Iteration can be expressed as *O(logn)* because it of its quadratic convergence rate. Each iteration essentially brings you twice as close to the answer than the previous iteration. However, we must also include the time it takes to calculate *f(x) / f'(x)*. This can be expressed as *F(n)*, where n is the number of digits of precision. Thus the actual time complexity of this algorithm is *O(logn F(n))* ("Newton's Method").

Issues:

There is no guarantee of convergence when using Newton's Iteration. Estimates may never converge to the correct answer to a certain precision for a variety of reasons. It may be that the initial estimate was too far off from the roots for the algorithm to work. A method of making a better initial guess is to use to apply the Intermediate Value Theorem. Given an interval [a,b], if the f(a) and f(b) are opposite signs (positive / negative), then there is guaranteed to be a root in that interval and so you can assign your guess as something in that interval.

Works Cited

Knill, Oliver. “Handouts.” *Math 1a, Spring 2012, Functions and Calculus, Harvard College/GSAS: 8434*, 2012, http://www.math.harvard.edu/~knill/teaching/math1a_2012/handouts.html.

“Newton's Method.” *Citizendium*, Citizendium, http://en.citizendium.org/wiki/Newton's_method.

Peng, Roger D. “Advanced Statistical Computing.” *Advanced Statistical Computing*, 17 July 2018, https://bookdown.org/rdpeng/advstatcomp/rates-of-convergence.html.

Salvy, Bruno. “CSE202 - Design and Analysis of Algorithms (2018-2019).” *Course: CSE202 - Design and Analysis of Algorithms (2018-2019),* https://moodle.polytechnique.fr/course/view.php?id=6782#section-4.

Weisstein, Eric W. "Newton's Method." *From MathWorld--A Wolfram Web Resource*. http://mathworld.wolfram.com/NewtonsMethod.html

Walls, Patrick. “Newton's Method.” *Mathematical Python*, 10 Sept. 2019, https://www.math.ubc.ca/~pwalls/math-python/roots-optimization/newton/.