CS100B Fall 1999

Mathematical Exposé on Newton's Method

David I. Schwartz

1. Goals

This report explains Newton's Method for root solving. A majority of this report reviews background concepts/formula/theory that all help explain the "gist" of Newton's Method. Students who need to refresh their memory of some underlying mathematics should thoroughly review this report. Also, anyone interested in discovering how to apply Java, Maple, and MATLAB to these concepts should check each section for scattered bits of code.

2. Functions

Recall some basic concepts about functions. Although many notions appear obvious, understanding Newton's Method requires a solid foundation of basic algebra.

2.1 The Basics

Let $y = f(x)$, where

- y is the ***dependent variable***, plotted on the vertical axis
- x is the ***independent variable***, plotted on the horizontal axis.

2.2 Plotting

For example, consider the plot of $f(x) = x^2 - 1$ in Figure 1.

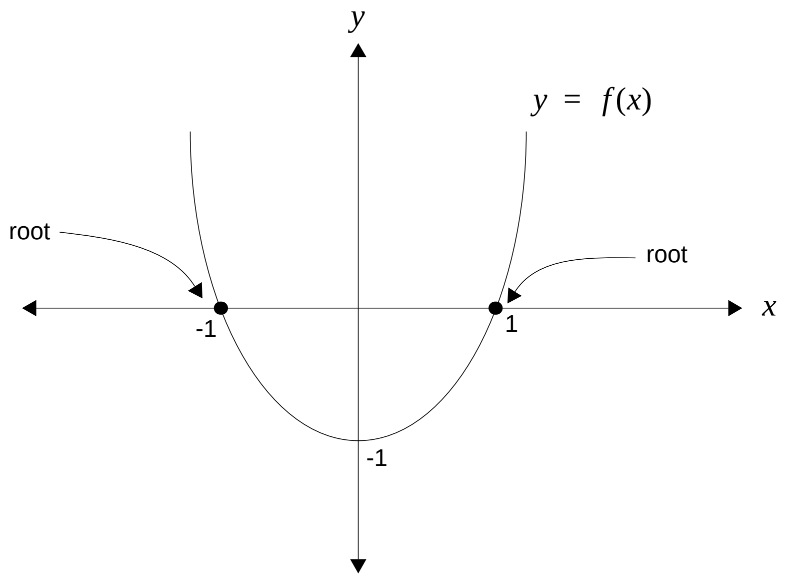

Figure 1: Plot of Function

How can you create this graph? Because the variable x is independent, you can pick arbitrary values of x. Given an arbitrary value of x, you would then calculate $f(x)$ from the polynomial $x^2 - 1$. For example, at $x = 2$, $f(x) = 2^2 - 1 = 3$. Thus, for $x = 2$, $y = f(2) = 3$. So, you plot the point $(x, y) = (2, 3)$ and, then, try other values of x.

2.3 Algorithm

Remember that y depends on x. So, when converting a function into computer code, you need to let the computer vary the independent variable. Each time the independent variable changes, you can let the computer calculate a new value for the dependent variable. For instance, to implement the example presented above in Section 2.2, follow these steps:

1. Declare your variables:
   - Let **`y`** be dependent.
   - Let **`x`** be independent.
2. Choose starting and stopping values for the dependent variable:
   - Choose **`start = -2`**.
   - Choose **`stop = 2`**.
3. Increment **`x`** and iterate for values of **`y`**:
   - Let **`delta_x = 0.1`**.
   - Compute **`y = Math.pow(x,2) - 1`**.

2.4 Computer Code

The following code implements the algorithm in Section 2.3. Note that **`Format.java`**, borrowed from CUCS Stationery, provides methods for printing formatted output:

```
public class plotting_function
{
   public static void main(String args[])
      {

      double x; // independent variable
      double y; // dependent variable

      double start = -2;   // starting value of x
      double stop  =  2;   // stopping value of x
      double inc   =  0.1; // increment of x

      // iterate for different values of y given increments of x
      for (x = start; x <= stop; x=x+inc) {
         y = Math.pow(x,2) - 1;
         Format.print(System.out,”%4.2f “,x);  //See Format.java
         Format.print(System.out,”%4.2f\n”,y); //See Format.java
      }
   }
}
```

NOTE: `Format.java` stationery will be explained in an upcoming handout. For now, please accept on faith!

3. Slope

Newton's Method relies on a thorough understanding of ***slope***, a measure of how a function changes.

3.1 Approximate

Assume a function $y = f(x)$ is continuous and smooth for any value of x. Pick two values of x, called x_1 and x_2, that have respective values $y_1 = f(x_1)$ and $y_2 = f(x_2)$, as shown in Figure 2. The following equation represents the approximate slope between Points 1 and 2:

$$\text{Slope} \approx \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1}. \tag{1}$$

In general, denote slope with the letter m.

3.2 Lines

If $f(x)$ is linear, then Equation 1 is exact. For example, the line $y = f(x) = x + 1$ is linear. Two formulas for the equation of a line use Equation 1 and should look familiar:

$$y = mx + b, \tag{2}$$

given y-intercept b, and

$$y - y_1 = m(x - x_1). \tag{3}$$

Equation 3 is also called the ***point-slope formula*** of a line.

3.3 Exact

Calculus provides an analytical method for finding slope. Consider the curve $y = f(x)$, shown in Figure 2. Find the slope, m, of Line $\overline{12}$ using Equation 1:

$$m_{12} = \frac{f(x_2) - f(x_1)}{x_2 - x_1} = \frac{f(x_1 + \Delta x) - f(x_1)}{(x_1 + \Delta x) - (x_1)} = \frac{f(x_1 + \Delta x) - f(x_1)}{\Delta x}. \tag{4}$$

Apply mathematical limits such that Δx "shrinks" as $\Delta x \to 0$. Then, the Line $\overline{12}$ approaches the tangent to Point 1:

$$m_{\text{Tangent}} = \lim_{\Delta x \to 0} \frac{f(x_1 + \Delta x) - f(x_1)}{\Delta x} \tag{5}$$

Equation 5 represents the formula for ***derivatives***. A derivative measures slope, the instantaneous rate of change of a function at a point. Note that derivatives exist only where a function is smooth and continuous.

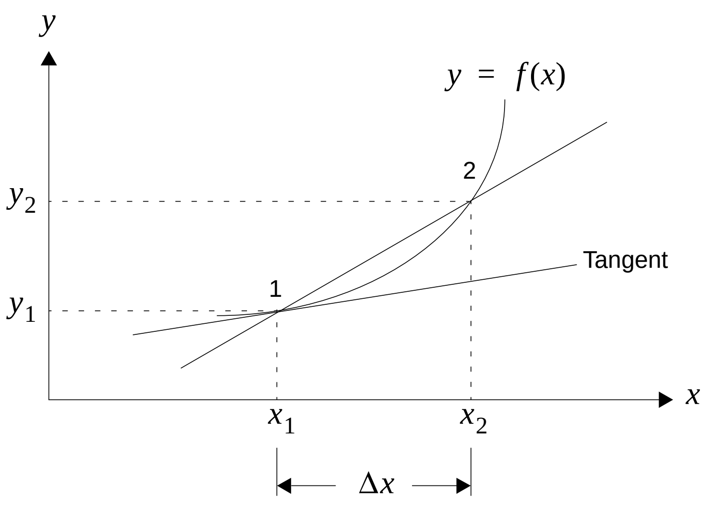

Figure 2: Finding Slopes

3.4 Notation

Using $\frac{\Delta y}{\Delta x}$ only approximates slope, but you can improve your accuracy. The ***first derivative*** of y with respect to x finds the exact, "instantaneous" slope anywhere on $f(x)$:

$$\text{First Derivative: } y' = f'(x) = \frac{dy}{dx} = \frac{d}{dx}f(x). \qquad \textbf{(6)}$$

Whereas Δ indicates a "large" change, d indicates an infinitesimal change in a variable. Thus, a derivative provides an exact slope at a point.

3.5 Implementation

Except when using a numerical method for computing derivatives, like finite differences, you should just work out the derivative by hand. You can then express the formula you derived using mathematical expressions.

For example, consider the polynomial function $y = f(x) = x^3 + 2x + 1$. The first derivative is $f'(x) = 3x^2 + 2$ using rules of differentiation. Using Java, you could enter the following code to compute the values of $f(x)$ and $f'(x)$ at $x = 2$:

```
double x = 2;                          // assign 2 to x
double y = Math.pow(x,3) + 2*x + 1;    // assign f(2) to y
double y_prime = 3*Math.pow(x,2) + 2; // assign f’(2) to y_prime
```

3.6 Mathematical Software

Would you prefer that computer do the derivative as well? Consult programs called Computer Algebra Systems (CAS), like Maple. For instance, the following Maple code will compute the derivative for you:

- Maple

```
> diff(x^3+2*x+1,x); Maple code: Do not enter the prompt >!
```

If you prefer MATLAB, the following code instructs MATLAB to activate its CAS, which happens to be Maple!

- MATLAB:

```
>> f = sym('x^3+2*x+1'); MATLAB code: Do not enter the prompt >>!
>> diff(f)
```

Both approaches produce the identical result of $3x^2 + 2$.

4. Roots

4.1 Equation

Sometimes functions are expressed in terms of an ***equation***, where $y = f(x)$. When $y = 0$, you can say $f(x) = 0$. If y has a constant value, you can still rearrange the equation to get a zero on one side. For example, $x^2 = 1$ is equivalent to $x^2 - 1 = 0$.

4.2 Assumptions

Let $y = f(x)$, where f a function of x. Assume that $f(x)$ is smooth and continuous, so you can rest assured $f(x)$ is differentiable "anywhere" you like. So, you can calculate $f'(x)$ at any value of x. Solving for a derivative, the slope at a point, will help find a root of $f(x)$. So, what's a *root*?

4.3 Real Roots

A particular value of x that will produce $f(x) = 0$ is called a ***root***. For instance, given $f(x) = x^2 - 1$, the values $x = 1$ and $x = -1$ produce 0, as shown in Figure 1. When roots have real values, sometimes the roots are called ***real roots***. Since $y = f(x) = 0$ when x is root, the plot crosses the *y*-axis. This notion, though a bit obvious, does explain why plotting a graph helps understand an equation's behavior.

4.4 Approximate Solution

Think about this simple statement: When x is a root, $y = f(x) = 0$. Without doing much math, you can try plotting a function to see where the function crosses the vertical axis. Refer to the code in Section 2.4. Try different starting and stopping values, and inspect the output for a change in signs.

NOTES: A better approach would involve either storing the two values of **x** just "before" and just "after" the sign of **y** changes. You could also send the **x** and **y** data to a file and read in the values into MATLAB to produce a plot. (A reminder: something we should enhance here.)

4.5 Imaginary Roots

Although not mandatory, sometimes imaginary roots will also produce 0 in a function. Recall that an ***imaginary number*** i is defined as follows:

$$i = \sqrt{-1}\,. \qquad \textbf{(7)}$$

Consider the forth-order polynomial $x^4 - 1$. The values $1, -1, i$, and, $-i$ are all roots. You can use both Maple and MATLAB to test these results:

- Maple:

```
> solve(x^4-1=0,x);
```

$$1, -1, I, -I$$

- MATLAB:

```
>> sym('x^4-1=0');
>> solve(f,'x')
ans =
[  1]
[ -1]
[  i]
[ -i]
```

In general, ignore potential imaginary roots and concentrate on finding real roots.

5. Newton's Method

Newton's Method is a quick, efficient approach that finds a root of a function. This section assumes that you understand concepts of functions, equations, plotting, slopes, and derivatives. If not, please review earlier sections in this report.

5.1 Using Tangents

Most numerical approaches involve some guessing. You might even get lucky! Assume that you need to find the root x_r of a function. Try the following steps assuming you have a differentiable function $f(x)$ with slope $f'(x)$:

1. Guess a value of x.
    - Call this value x_1, your first "guess" of x.
    - Use the code in Section 2.4 to provide a "close" answer.
    - Check if you "got lucky." If not, continue on.…
2. Consider the tangent to the function $f(x_1)$ at x_1.
    - Compute the *x*-intercept of the tangent, the position where the tangent crosses the horizontal axis.
    - You can use the point-slope formula for the *tangent line* given $f(x)$ and $f'(x)$:

$$y - f(x_1) = f'(x_1)(x - x_1) \qquad \textbf{(8)}$$

    - The tangent line crosses the *x*-axis at the point $(x_2, 0)$, as shown in Figure 3. Thus, $y = 0$ when $x = x_2$. From the point-slope formula, you can then say

$$0 - f(x_1) = f'(x_1)(x_2 - x_1) \tag{9}$$

for the tangent line. Remember: Equation 9 represents the tangent line drawn from your first guess, x_1, but *not* the function $f(x)$!

- Rearranging Equation 9 produces the following result that finds x_2:

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}. \tag{10}$$

3. Repeat Step 2, as shown in Figure 3.
   - x_2 becomes the new "guess," resembling x_1.
   - Solve for x_3 by Equation 10, using a new tangent line drawn at x_2:

$$x_3 = x_2 - \frac{f(x_2)}{f'(x_2)}. \tag{11}$$

4. Stop iterating when your "result" is "close enough:"
   - The previous and new value of x differ less than a given tolerance ε.
   - The value of $f(x)$ is close to zero within a given tolerance ε.

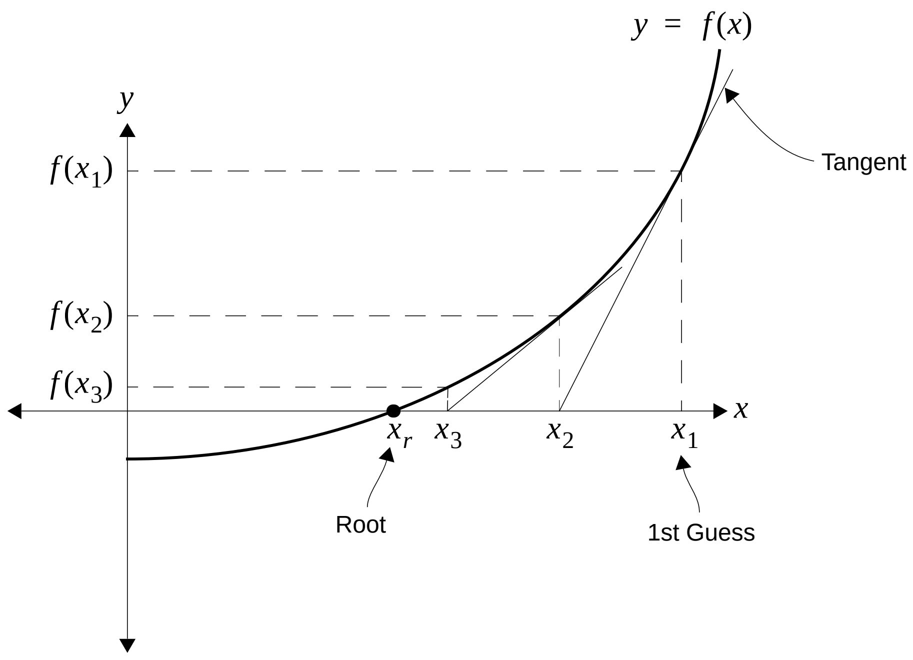

Figure 3: Newton's Method

5.2 Algorithm

The following algorithm is modified from Programming Assignment 2. This version enforces tolerance in $f(x)$. Assume that the function $f(x)$ is assigned to the variable **`f_of_x`**. Also, assume that the derivative $f'(x)$ is assigned to **`deriv`**. Refer to Section 3.5 for more details on coding these functions:

- Pick an initial value **`x`**.
- Calculate **`f_of_x`**.
- While $|f(x) - 0| \leq \varepsilon$, iterate as follows:
    - Compute the value of $f(x)$. Assign the result to **`f_of_x`**.
    - Compute the value of $f'(x)$. Assign the result to **`deriv`**.
    - Compute the new value of x using **`x = x - f_of_x/deriv`**.
    - Compute the new value of $f(x)$.

5.3 Notes

- Stopping condition: Checking either x or $f(x)$ will work, though checking $f(x)$ might provide a more accurate result. For instance, if a root is along a "steep" curve, small changes in x make "big" changes in $f(x)$.
- Convergence: Newton's Method is not perfect! If the slope of the function is zero or if there are multiple roots, Newton's Method is insufficient. More advanced algorithms in numerical analysis account for these facts and improve this basic algorithm.