Newton’s method convergence speed

Newton’s method [0] is one of those special numerical methods which is easy to grasp, yet very, very fast. On the right conditions, it’s rate of convergenge is brutally fast compared to other root finding algorithms, such as gradient descent[1] or bissection[2].

Deduction

Newton’s method deduction is quite straight forward. It starts by acknowledging that many functions can be linearly approximated about $x = x_i$ as

$$f(x) \approx f(x_i) + \frac{\partial f}{\partial x}(x_i)(x - x_i)$$

Suppose that x_r is a root, then using the linear approximation we can write

$$f(x_r) = 0 \approx f(x_i) + \frac{\partial f}{\partial x}(x_i)(x_r - x_i)$$

Rearranging this expression, we obtain the following expression for x_r

$$x_r \approx x_i - \frac{f(x_i)}{\frac{\partial f}{\partial x}(x_i)}$$

In linear cases ($f(x) = a + bx, b \neq 0$), the approximation sign can be replaced with the equal sign. In the other cases we will use the approximation of x_r as x_{i+1} and create a new linear approximation around x_{i+1} that point to generate the new approximation of x_r. This sequence can be encoded as

$$x_{i+1} = x_i - \frac{f(x_i)}{\frac{\partial f}{\partial x}(x_i)}$$

After some j iterations, x_{i+j} should be very close to x_r.

Graphically, what is happening is the following

(Fazer 2 ou 3 gráficos da iteração) (0: expansão linear) (1: obtenção de x_i+1) (2: nova expansão linear)

Speed

So, how fast it is, really?

To analyse the rate of convergence, we will use the error $\varepsilon_i = x_i - x_r$ and calculate it’s rate of convergence $R_i = \frac{\varepsilon_{i+1}}{\varepsilon_i}$. Lower values of R_i give us a faster convergence ???????. We can just run this algorithm for some function $f(x)$ and see how the error reduces with each new iteration. Or, we can use an abstract approach, giving us a more general result.

Suppose that our function can be represented by it's Taylor expansion around the root x_r

$$f(x) = \sum_{k=0}^{\infty} a_k \frac{(x - x_r)^k}{k!}$$

Because we are expanding this sequence around x_r, the only a_k that will survive is a_0. But $f(x_r) = 0$, then $a_0 = 0$.

Suppose that x_i is close to x_r, such that $\varepsilon_i = x_i - x_r \approx 0$. Using $x_i = x_r + \varepsilon_i$, Newton's algorithm iteration can be rewritten as

$$\varepsilon_{i+1} = \varepsilon_i - \frac{\sum_{k=1}^{\infty} a_k \frac{\varepsilon_i^k}{k!}}{\sum_{k=1}^{\infty} a_k \frac{k}{\varepsilon_i} \frac{\varepsilon_i^k}{k!}} = \varepsilon_i \frac{\sum_{k=1}^{\infty} a_k (k-1) \frac{\varepsilon_i^k}{k!}}{\sum_{k=1}^{\infty} a_k k \frac{\varepsilon_i^k}{k!}}$$

Now we have to study this fraction. Let us define the first non-zero term (of a sum) as theLet us define the first non-zero term (of a sum) as the ... Given that ε_i is almost 0, the a_k with lowest k that is non zero, will survive in the fraction. But there is a catch. The $k = 1$ term in the numerator is 0. So the surviving term will be the first non-zero a_k with $k > 1$, while in the denominator is $k > 0$.

Let's study cases.

In case $a_1 = 0$, let us assume that a_j is the first non-zero term. Then we can aproximate ε_{i+1} as

$$\varepsilon_{i+1} \approx \varepsilon_i \frac{a_j (j-1) \frac{\varepsilon_i^j}{j!}}{a_j j \frac{\varepsilon_i^j}{j!}} = \varepsilon_i \frac{j-1}{j}$$

We see that $R_i = \frac{j-1}{j}$, which is not so bad for low j, but not so good for high j, almost stalling. In fact

$$\varepsilon_i \approx \varepsilon_0 \left(\frac{j-1}{j} \right)^i$$

Therefore, to reach a given error E, we need a number of steps

$$i \approx \frac{\log(\frac{E}{\varepsilon_0})}{\log(\frac{j-1}{j})} \sim \log \left(\frac{1}{E} \right)$$

In case $a_1 \neq 0$, let us assume that a_j is the first non-zero besides a_1. Then we can aproximate ε_{i+1} as

$$\varepsilon_{i+1} \approx \varepsilon_i \frac{a_j (j-1) \frac{\varepsilon_i^j}{j!}}{a_1 \varepsilon_i} = \varepsilon_i^j \frac{a_j (j-1)}{a_1 j!} = \varepsilon_i^j c_j$$

Therefore

$$e_1 = e_0^j c_j$$

$$e_2 = e_1^j c_j = (e_0^j c_j)^j c_j = e_0^{j^2} c_j^{j+1}$$

$$e_3 = (e_0^{j^2} c_j^{j+1})^j c_j =$$

The algorithm starts with a raw guess, hopefully near the root value, we call it x_0

[0] https://en.wikipedia.org/wiki/Newtons_method

[1]

[2]