Chapter 3

Big Oh Notation

In dealing with functions that vary on the number of steps or the step size, it is common to arrive at a series whereby we only want to consider the most significant terms(those that contribute the most) and ignore the rest. Let's consider this in three cases:

3.1 Quantity $x >> 1$

If x is much greater than 1, the most significant term is the term with the highest order. For example, if $y = x^3 + 3x^2 + 1000x$, the highest order term x^3 completely dominates the polynomial. To see this, take $x \approx 10^{10}$ and observe the contribution from each term:

$$x^3 \approx 10^{30} \tag{3.1}$$
$$3x^2 \approx 10^{20} \tag{3.2}$$
$$1000x \approx 10^{13} \tag{3.3}$$

Because of this, we usually say $y = O(x^3)$, or y is on the order of x^3 for large x.

The most prominent use of this case is calculating algorithm runtimes, and typical use would be to say (for example) that a matrix solve on an $n \times n$ matrix was $O(n^3)$ or the runtime for finding the maximum value in a $n \times 1$ vector is $O(n)$. Another example is that usually we assume a 'for' loop will take $O(n)$ time, where n is the number of items it will process.

3.2 Quantity $x << 1$

In the opposite case where x is small, the most significant term is the lowest order term. For example, in the same equation $y = x^3 + 3x^2 + 1000x$, we can say that $y = O(x)$. In this case, if we took $x \approx 10^{-10}$, then

$$x^3 \approx 10^{-30} \tag{3.4}$$
$$3x^2 \approx 10^{-20} \tag{3.5}$$
$$1000x \approx 10^{-7} \tag{3.6}$$

showing that the largest contribution came from the $1000x$ term.

This case is typically used in the error analysis of numerical schemes, because this is when x is typically small. To see another example and rewrite a Taylor Series approximation in this notation,

$$f(x_0 + \Delta x) = \sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!} + O((\Delta x)^{n+1}) \tag{3.7}$$

3.3 Quantity $x \approx$ some constant

If our quantity y does not depend on the size of the numbers we are considering, then we might say $y = O(1)$. This is not to say that is near one, just that y does not have a strong dependance on x, so it's behavior is not dominated by it.

Some typical examples of this are the computer time required to access a single element in a list or to perform one multiplication of two real numbers, since both of these will take a relatively constant amount of time.

3.4 General Notes

Hopefully, these examples will provide a base intuition for thinking about this notation, which is likely to be more useful than the actual formula any of this can be derived from. However, for the sake of completely, this entire discussion is equivalent to the following definitions:

$$f(x) = O(g(x)) \quad \equiv f(x) \leq Cg(x) \text{ as } x \to \infty \tag{3.8}$$

$$f(x) = O(g(x)) \quad \equiv f(x) \leq Cg(x) \text{ as } x \to 0 \tag{3.9}$$

$$\text{for some values of C} \tag{3.10}$$

A more precise definition is crucial for determining the order of nonpolynomial functions. A prominent example of this is the Fast Fourier Transform, which has the runtime $O(n \log_2 n)$, a huge performance increase over typical transform runtimes of $O(n^2)$.

It is also important to note that Big-Oh notation provides upper bounds on runtimes or errors. For example, for any algorithm with an $O(n^2)$ runtime, it is also mathematically true to state that the algorithm is $O(n^{100})$. Such a gross overestimate of algorithm run time is unlikely to generate interest in an algorithm, however, so it is important to use the 'tightest' bounds that you can justify using.

It is also important to note that as a matter of convention, the order notation is never written as $O(n^k) = f(n)$, but always $f(n) = O(n^k)$. This arises from the alternative(and perhaps better!) notation of considering $O(g(n))$ as the set of all functions satisfying 3.8 for $g(n)$, then saying that your function $f(n) \in O(g(n))$.