# B. Asymptotics and Computational Cost

We introduce Big-O, little-o and asymptotic notation and see how they can be used to describe computational cost.

1. Asymptotics as $n \to \infty$
2. Asymptotics as $x \to x_0$
3. Computational cost

## 1. Asymptotics as $n \to \infty$

Big-O, little-o, and "asymptotic to" are used to describe behaviour of functions at infinity.

**Definition 1 (Big-O)**

$$f(n) = O(\phi(n)) \qquad (\text{as } n \to \infty)$$

means $\left|\frac{f(n)}{\phi(n)}\right|$ is bounded for sufficiently large $n$. That is, there exist constants $C$ and $N_0$ such that, for all $n \geq N_0$, $|\frac{f(n)}{\phi(n)}| \leq C$.

**Definition 2 (little-O)**

$$f(n) = o(\phi(n)) \qquad (\text{as } n \to \infty)$$

means $\lim_{n \to \infty} \frac{f(n)}{\phi(n)} = 0.$

**Definition 3 (asymptotic to)**

$$f(n) \sim \phi(n) \qquad (\text{as } n \to \infty)$$

means $\lim_{n \to \infty} \frac{f(n)}{\phi(n)} = 1.$

**Examples**

1. $

{\cos n \over n^2 -1} = O(n^{-2}) $as \left|\frac{\frac{\cos n}{n^2-1}}{n^{-2}}\right| \leq \left|\frac{n^2}{n^2-1}\right| \leq 2 for$n \geq N_0 = 2$.

2. $

\log n = o(n) $as$ \lim_{n → ∞} {\log n \over n} = 0. $

3. $

n^2 + 1 ~ n^2 $as$ {n^2 +1 \over n^2} → 1. $

Note we sometimes write $f(O(\phi(n)))$ for a function of the form $f(g(n))$ such that $g(n) = O(\phi(n))$.

## Rules

We have some simple algebraic rules:

**Proposition 1 (Big-O rules)**

$$\begin{aligned} O(\phi(n))O(\psi(n)) &= O(\phi(n)\psi(n)) \qquad (\text{as } n \to \infty) \\ O(\phi(n)) + O(\psi(n)) &= O(|\phi(n)| + |\psi(n)|) \qquad (\text{as } n \to \infty). \end{aligned}$$

# 2. Asymptotics as $x \to x_0$

We also have Big-O, little-o and "asymptotic to" at a point:

**Definition 4 (Big-O)**

$$f(x) = O(\phi(x)) \qquad (\text{as } x \to x_0)$$

means $|\frac{f(x)}{\phi(x)}|$ is bounded in a neighbourhood of $x_0$. That is, there exist constants $C$ and $r$ such that, for all $0 \leq |x - x_0| \leq r$, $|\frac{f(x)}{\phi(x)}| \leq C$.

**Definition 5 (little-O)**

$$f(x) = o(\phi(x)) \qquad (\text{as } x \to x_0)$$

means $\lim_{x \to x_0} \frac{f(x)}{\phi(x)} = 0$.

**Definition 6 (asymptotic to)**

$$f(x) \sim \phi(x) \qquad (\text{as } x \to x_0)$$

means $\lim_{x \to x_0} \frac{f(x)}{\phi(x)} = 1$.

**Example**

$$\exp x = 1 + x + O(x^2) \qquad \text{as } x \to 0$$

since $\exp x = 1 + x + \frac{\exp t}{2} x^2$ for some $t \in [0, x]$ and

$$\left| \frac{\frac{\exp t}{2} x^2}{x^2} \right| \leq \frac{3}{2}$$

provided $x \leq 1$.

## 3. Computational cost

We will use Big-O notation to describe the computational cost of algorithms. Consider the following simple sum

$$\sum_{k=1}^{n} x_k^2$$

which we might implement as:

In [1]:
```
function sumsq(x)
    n = length(x)
    ret = 0.0
    for k = 1:n
        ret = ret + x[k]^2
    end
    ret
end

n = 100
x = randn(n)
sumsq(x)
```

Out[1]: 90.95882254617737

Each step of this algorithm consists of one memory look-up (`z = x[k]`), one multiplication (`w = z*z`) and one addition (`ret = ret + w`). We will ignore the memory look-up in the following discussion. The number of CPU operations per step is therefore 2 (the addition and multiplication). Thus the total number of CPU operations is $2n$. But the constant $2$ here is misleading: we didn't count the memory look-up, thus it is more sensible to just talk about the asymptotic complexity, that is, the *computational cost* is $O(n)$.

Now consider a double sum like:

$$\sum_{k=1}^{n} \sum_{j=1}^{k} x_j^2$$

which we might implement as:

In [2]:
```
function sumsq2(x)
    n = length(x)
    ret = 0.0
    for k = 1:n
        for j = 1:k
            ret = ret + x[j]^2
        end
    end
    ret
end
```

```julia
n = 100
x = randn(n)
sumsq2(x)
```

Out[2]: 5377.916656174812

Now the inner loop is $O(1)$ operations (we don't try to count the precise number), which we do $k$ times for $O(k)$ operations as $k \to \infty$. The outer loop therefore takes

$$\sum_{k=1}^{n} O(k) = O\left(\sum_{k=1}^{n} k\right) = O\left(\frac{n(n+1)}{2}\right) = O(n^2)$$

operations.