Order notation

We say that

\[ f(n) = O(g(n)) \]

read as “\( f \) is big-oh of \( g \)” or “\( f \) is of order \( g \)” if there is a positive constant \( C \) such that

\[ |f(n)| \leq C|g(n)| \]

for all \( n \) sufficiently large. For example,

\[ 2n^3 + 3n^2 + n = O(n^3) \]

because as \( n \) becomes large, the terms of order lower than \( n^3 \) become relatively insignificant. For an accurate estimate, we are interested in the behavior as some quantity \( h \), such as a “step size” or “mesh spacing” becomes very small. We say that

\[ f(h) = O(g(h)) \]

if there is a positive constant \( C \) such that

\[ |f(h)| \leq C|g(h)| \]

for all \( h \) sufficiently small. For example,

\[ \frac{1}{1-h} = 1 + h + h^2 + h^3 + \ldots = 1 + h + O(h^2) \]

because as \( h \) becomes small, the omitted terms beyond \( h^2 \) become relatively insignificant. Note that the two definitions are equivalent if \( h = 1/n \).
<table>
<thead>
<tr>
<th>$x$ (decimal notation)</th>
<th>$x$ (scientific notation)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2012</td>
<td>$2.012 \times 10^3$</td>
</tr>
<tr>
<td>412</td>
<td>$4.12 \times 10^2$</td>
</tr>
<tr>
<td>3.14</td>
<td>$3.14 \times 10^0$</td>
</tr>
<tr>
<td>0.000789</td>
<td>$7.89 \times 10^{-4}$</td>
</tr>
<tr>
<td>0.2091</td>
<td>$2.091 \times 10^{-1}$</td>
</tr>
</tbody>
</table>

### How are numbers stored on the computer?

First, we shall review the concept of “scientific notation”, which will give us some helpful insights. For any decimal number $x$ (we assume that $x$ is a terminating decimal number, with finite non-zero digits) we can write

$$x = a \times 10^b, \quad \text{where} \quad 1 \leq |a| < 10$$

**Exception:** When $x = 0$, we simply set $a = b = 0$. For example:

Every decimal (or Base 10) number can be written as

$$a_k a_{k-1} \ldots a_2 a_1 a_0.a_{-1} a_{-2} a_{-3} \ldots a_{-m} = \sum_{i=-m}^{k} a_i 10^i$$

For example

<table>
<thead>
<tr>
<th>$x$</th>
<th>$a_3$</th>
<th>$a_2$</th>
<th>$a_1$</th>
<th>$a_0.$</th>
<th>$a_{-1}$</th>
<th>$a_{-2}$</th>
<th>$a_{-3}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.14</td>
<td>3</td>
<td></td>
<td>1</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.037</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Binary (Base 2) fractional numbers can be written as

$$b_k b_{k-1} \ldots b_2 b_1 b_0.b_{-1} b_{-2} b_{-3} \ldots b_{-m} = \sum_{i=-m}^{k} b_i 2^i$$

where every digit $b_i$ is now only allowed to equal 0 or 1. For example

- $5.75 = 4 + 1 + 0.5 + 0.25 = 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^{-1} + 1 \times 2^{-2} = 101.11_2$
- $17.5 = 16 + 1 + 0.5 = 1 \times 2^4 + 1 \times 2^0 + 1 \times 2^{-1} = 10001.1_2$

Note that certain numbers that are finite (terminating) decimals actually are periodic in binary, e.g.

$$0.4_{(10)} = 0.011001100110_2 = 0.011001110011_2$$ (1)
Machine numbers

This is an abbreviation for binary floating point numbers. The numbers stored on the computer are essentially binary numbers, in scientific notation $x = \pm a \times 2^b$. Here, $a$ is called the mantissa and $b$ the exponent. We also follow the convention that $1 \leq a < 2$; the idea is that for any number $x$, we can always divide it by an appropriate power of 2, such that the result will be within $[1, 2)$. For example:

$$x = 5_{(10)} = 1.25_{(10)} \times 2^2 = 1.01_{(2)} \times 2^2$$

Thus, a machine number is stored as:

$$x = \pm 1.a_1a_2\ldots a_{k-1}a_k \times 2^b$$

- In **single precision** we store $k = 23$ binary digits, and the exponent $b$ ranges between $-126 \leq b \leq 127$. The largest number we can thus represent is $(2 - 2^{-23}) \times 2^{127} \approx 3.4 \times 10^{38}$.

- In **double precision** we store $k = 52$ binary digits, and the exponent $b$ ranges between $-1022 \leq b \leq 1023$. The largest number we can thus represent is $(2 - 2^{-52}) \times 2^{1023} \approx 1.8 \times 10^{308}$.

In other words, single precision provides 23 binary significant digits. In order to translate it to familiar decimal terms we note that $2^{10} \approx 10^3$, thus 10 binary significant digits are roughly equivalent to 3 decimal significant digits. Using this, we can say that single precision provides approximately 7 decimal significant digits, while double precision offers slightly more than 15.

Absolute and relative error

As discussed in the previous lecture, all computations on a computer are approximate by nature, due to limited precision on the computer. As a consequence, we have to tolerate some amount of error in our computation. In order to better understand errors in computation, we use the absolute and relative error measures. Let $q$ denote the exact (analytic) quantity that we expect out of a given computation, and $\hat{q}$ denote the (likely compromised) value actually generated by the computer.

The **absolute error** is $e = |q - \hat{q}|$. This is useful when we want to frame the result within a certain interval, since $e \leq \delta$ implies $q \in [\hat{q} - \delta, \hat{q} + \delta]$.

The **relative error** is $e = |q - \hat{q}|/|q|$. The result may be expressed as a percentile and is useful when we want to assess the error relative to the value of the exact quantity. For example, an absolute value of $10^{-3}$ may be insignificant when the intended value of $q$ is in the order of $10^6$, but would be very severe if $q \approx 10^{-2}$.
Rounding and truncation

When storing a number on the computer, if the number happens to contain more digits than it is possible to represent via a machine number, an approximation is made via *rounding* or *truncation*. When using truncated results, the machine number is constructed by simply discarding significant digits that cannot be stored; rounding approximates a quantity with the closest machine-precision number. For example, when approximating \( \pi = 3.14159265 \ldots \) to 4 decimal significant digits, truncation would give \( \pi \approx 3.1415 \) while the rounded result would be \( \pi \approx 3.1416 \). Rounding and truncation are similarly defined for binary numbers, for example, \( x = 0.1011011101110 \ldots_{(2)} \) would be approximated to 5 binary significant digits as \( x \approx 0.10110_{(2)} \) using truncation, and \( x \approx 0.10111_{(2)} \) when rounded.