Computer arithmetic and errors

Francesco Marchetti

Fundamentals of Computational Mathematics

Summary

Math can be inexact here!

In some situations, when working with a PC, we may face the somehow embarassing situation where

$$1 + \epsilon = 1$$

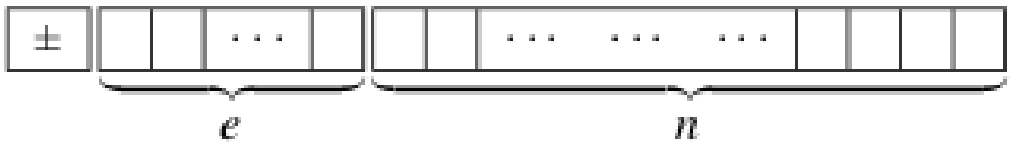
and $\epsilon \neq 0$. This is because not all numbers can be represented exactly in a computer, and arithmetic mistakes are a consequence. The information in a **single-precision** PC is built upon vectors composed by "0/1"32 bits (4 bytes). A large number of integers can be thus represented exactly, in base 2.

Real numbers

The situation with real numbers $x \neq 0$ is more complicated. The representation of x is structured as

$$x = \pm n \cdot b^e,$$

where n is the **mantissa**, b is the **base** (always 2) and e is the exponent. This representation is also called **floating point**.



It is important to notice that, no matter the number of bits employed, a limited bits are assigned for e and n, therefore not all real numbers can be represented.

The most common setting

Ordinary modern personal computers use a 64-bit architecture (**double-precision**). A number of $n = 53$ bits are dedicated to the mantissa (actually 52 since one bit is *hidden*, as the first more significant digit is set to be always 1 and does not need to be stored), $e = 11$ for the exponent and one bit for the sign.
The smallest representable number is $m \approx 2.22 \cdot 10^{-308}$, while the biggest is $M \approx 1.79 \cdot 10^{308}$. There are some asymmetries due to some special representations, we are not going to deepen this topic. Also, some softwares can extend beyond m and M in some situations.

We remark that the density of representable numbers is larger near 0 and gets smaller and smaller as numbers get larger.

Machine precision

The **machine precision** ϵ is the smallest machine number such that

$$1 + \epsilon > 1$$

in floating point arithmetic (i.e., in the PC). In a 64-bit setting, $\epsilon \approx 2.2 \cdot 10^{-16}$. It is really important to take this into account when performing numerical experiments, because it implies that digits beyond the 16th are meaningless.

Note that floating point arithmetic is weird: sum and multiplication are commutative, but sometimes not associative.

Errors!

Two undesired situations follow:

- If a representable number is smaller than m, we get an **underflow** that results in a 0 value.
- If a representable number is greater than M, we get an **overflow** that results in a `Inf` value.

We can measure the precision of an approximation by computing for example:

- The **absolute error** between x and an approximation $\hat{x}$, which is $|x - \hat{x}|$.
- The **relative error**, which is $|x - \hat{x}|/|x|$. If $|x| \approx 0$, we can use the mixed formulation $|x - \hat{x}|/(|x| + 1)$

Other type of errors can be considered too. Moreover, **rounding** a number is always more precise than **truncating**!

The Vancouver stock exchange index

Topic: rounding vs truncation

The index was initialized at 1000 in January 1982. On each trade, the update was carried out by also truncating to three decimal places. After thousands of daily trades, in late November 1983 the index value was 524.811. Correcting the evolution of the index by using rounding in place of truncation led to a value of 1098.892.

The Ariane 5 explosion

Topic: overflow

The ESA Ariane 5 rocket was first launched on June 4th 1996. Just after launch, the inertial navigation system started to send data in 64-bit to the main computer that was set to work in 16-bit (re-usage of old software from the Ariane 4 programme). As a consequence, after a huge number was passed to the main computer (horizontal velocity), this resulted in overflow, thus bad monitoring and calibration of the trajectory and finally the explosion of the rocket. The resulting loss was 370 million dollars!

The Patriot Missile malfunctioning

Topic: accumulation of rounding errors

February 25th 1991, Gulf War. An American Patriot Missile battery in Dharan, Saudi Arabia, failed to track and intercept an incoming Iraqi Scud missile. This was due the propagation of small rounding errors in the internal clock time of the software. 28 soldiers were killed, plus more than 100 other people injured.

Instability vs ill-conditioning

There are two important concepts that can be confused but are different:

- **Instability**: the situation where rounding errors are not just propagated, but also amplified by some algorithm.
- **Ill-conditioning**: a problem is ill-conditioned when small changes (perturbations) of data, or in general of the problem setting, generate large changes in the solution.

It is important to observe that numerical instability is something that depends on the algorithm that we consider, therefore we can often do something to improve the situations. On the other hand, ill-conditioning il related to the nature of the problem, no matter the algorithm we choose to deal with it. And a lot of concrete problems are ill-conditioned, of course!

Instability: an example

We all know that the equation $ax^2 + bx + c = 0$ may admit two solutions

$$x_1 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}, \quad x_2 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}.$$

1: $\Delta = \sqrt{b^2 - 4ac}$
2: $x_1 = (-b - \Delta)/(2a)$
3: $x_2 = (-b + \Delta)/(2a)$

With $a = 1$, $c = 2$ and using 5 digits of precision, we have for different values of b:

b	Δ	float (Δ)	float (x_2)	x_2	$\frac{\lvert \text{float}(x_2) - x_2 \rvert}{\lvert x_2 \rvert + 1}$
5.2123	4.378135	4.3781	−0.41708	−0.4170822	1.55×10^{-6}
121.23	121.197	121.20	−0.01500	−0.0164998	1.47×10^{-3}
1212.3	1212.2967	1212.3	0	−0.001649757	Catastrophic cancelation

Note that $\text{float}(x_2) = \big(-b + \text{float}(\Delta)\big)/(2a)$

Instability: an example... and a solution

Alternatively, we can use the relation $x_1 x_2 = c/a$, and the following algorithm.

1: $\Delta = \sqrt{b^2 - 4ac}$
2: **if** $b < 0$ **then**
3: $\quad x_1 = (-b + \Delta)/(2a)$
4: **else**
5: $\quad x_1 = (-b - \Delta)/(2a)$
6: **end if**
7: $x_2 = c/(a\, x_1)$

With this idea, the computation of the solutions of second-order equation is stable (why?). The messages here are:

- Good theoretical tools do not correspond necessarily to good computational strategies (and viceversa!)
- Instability is something we need to take into account even when tasks are really simple.

Ill-conditioning: an example

Consider the linear system $A\boldsymbol{x} = \boldsymbol{b}$, with

$$A = \begin{pmatrix} 0.780 & 0.563 \\ 0.913 & 0.659 \end{pmatrix}, \quad \boldsymbol{b} = \begin{pmatrix} 0.217 \\ 0.254 \end{pmatrix},$$

whose solution is $\boldsymbol{x} = (1, -1)^T$. If we add a perturbation matrix

$$E = \begin{pmatrix} 0.001 & 0.001 \\ -0.002 & -0.001 \end{pmatrix},$$

and we consider the perturbated system $(A + E)\boldsymbol{x} = \boldsymbol{b}$, the new solution is $\boldsymbol{x} = (-5, 7.3085)^T$ (why is it so different?). This is an example of ill-conditioning, dealing with this might be harmful.

Condition number

Sometimes, we want to quantify the conditioning of our setting, i.e., the sensitivity of the solution of a problem $f(x)$ w.r.t. noise. In general, the conditioning of a (differentiable) function f with respect to input data x is

$$\mathrm{cond}(f)(x) = \frac{|xf'(x)|}{|f(x)|}.$$

When facing a linear system $Ax = b$, the conditioning of the linear system can be computed as

$$\kappa(A) = \|A\|\|A^{-1}\|,$$

where $\|\cdot\|$ is a matrix norm (we will come back to this!)
The condition number can be interpreted as an upper bound for the amount of inaccuracy that may result in the solution.

Computational complexity

Besides the stability, a crucial aspect related to an algorithm is the **computational complexity** (or computational time), that is, the number of elementary machine operations required to carry out the algorithm.
It is very common to indicate the order of the number of operations using a big-O notation, without being that precise. For example, one can say that an algorithm is $\mathcal{O}(n^2)$ is a number proportional to n^2 operations are required (e.g., $3n^2 + n$).

Good algorithms (maybe not that good but still acceptable) have a polynomial execution time with respect to n. On the other hand, algorithms that possess an exponential or factorial complexity are not usable.