Stability of algorithms

Problem: Is our algorithm (using floating point) going to compute a good approximation of the answer?

Related to sensitivity / conditioning but different. Depends on the details of the computation.

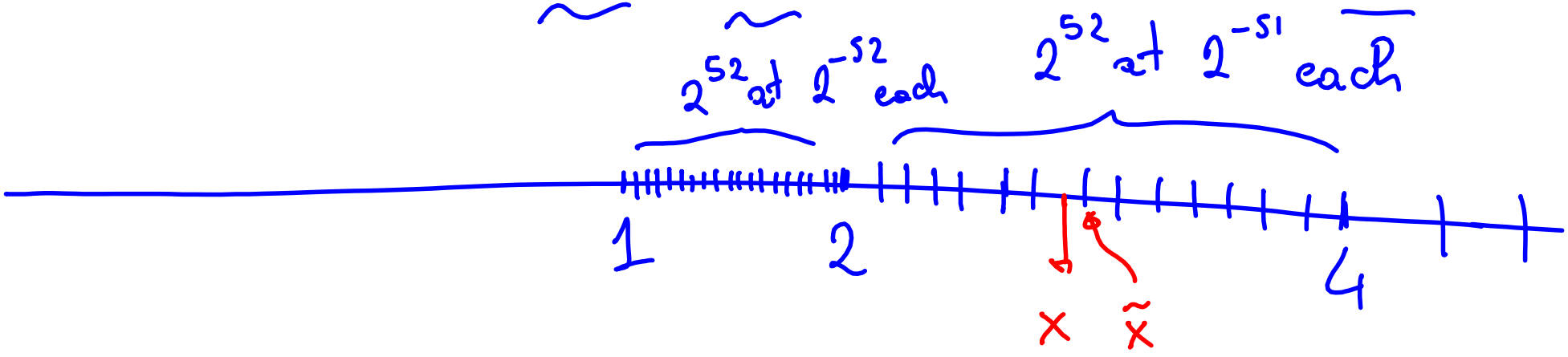
Floating point numbers in a nutshell

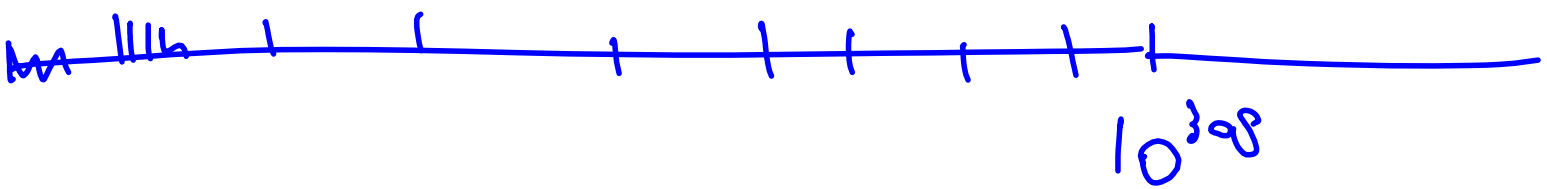
Floating point numbers are numbers in base-2 scientific (exponential) notation.

`double` (64-bit numbers):

$$\pm 1.\underbrace{01001011101\ldots101}_{52\text{ digits}} \cdot 2^{\pm\ \underbrace{101\ldots01}_{10\text{ digits}}}.$$

Plus 'error codes' like `Inf` and `NaN`, and technical details like `-0`.





Representation error

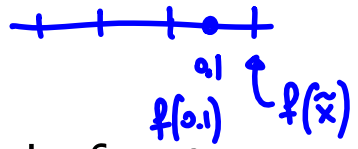
There are non-representable numbers, even simple ones such as 0.1.

There are 2^{52} floating point numbers between 1 and 2, spaced by $2^{-52} \approx 2 \cdot 10^{-16}$.
There are 2^{52} floating point numbers between 2 and 4, spaced by 2^{-51} each...

For each real $x \in \pm[10^{-308}, 10^{308}]$, there is an exactly representable number $\tilde{x}$ such that $\frac{|\tilde{x}-x|}{|x|} \leq \mathbf{u}$, with $\mathbf{u} = 2^{-52} \approx 2 \cdot 10^{-16}$.

$\frac{|\tilde{x}-x|}{|x|} \leq u \qquad u = 2^{-52} \approx 2 \cdot 10^{-16}$

Intrinsic error



Problem given code for `function y = f(x)` (for instance, $f(x) = \frac{x^2+1}{2x+5.5}$) am I going to get out of the computer the exact value of $f(0.1)$?

Answer: You can't even ask the computer to compute it!
The closest you can ask is $f(\tilde{x})$, where $\tilde{x}$ is the closest machine number to $x = 0.1$.

How far apart are $\tilde{y} = f(\tilde{x})$ and $y = f(x)$? That's a job for the condition number:

$$\frac{|\tilde{y} - y|}{|y|} \leq \kappa_{rel}(f, x)\frac{|\tilde{x} - x|}{|x|} + o\left(\frac{|\tilde{x} - x|}{|x|}\right) \approx u^2 \sim 10^{-32}$$

$$\leq \kappa_{rel}(f, x)\mathbf{u} + o(\mathbf{u}).$$

$$\frac{\frac{\|\tilde{y}-y\|}{\|y\|}}{\frac{\|\tilde{x}-x\|}{\|x\|}} \leq \max_{\frac{\|\tilde{x}-x\|}{\|x\|} \leq u} \frac{\frac{\|\tilde{y}-y\|}{\|y\|}}{\frac{\|\tilde{x}-x\|}{\|x\|}} = \kappa_{rel}(f,x) + o(1)$$

when $u \to 0$

Stability analysis

Apart from lucky cases (e.g., all intermediate numbers are exactly representable), you can't expect to compute $y = f(x)$ with better (relative) accuracy than $\kappa_{rel}(f, x)$.

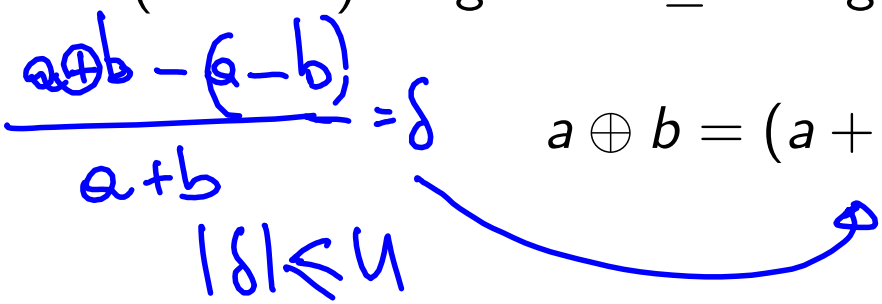
Still, some algorithms can be better than others. Case in point: that example with linear least squares.

How do we analyze the accuracy obtained by an algorithm?

With lots of tedious computations. . .

Operations on a computer produce the exact result + an error of (relative) magnitude $\leq \mathbf{u}$: e.g.,

$$a \oplus b = (a + b)(1 + \delta), \quad |\delta| \leq \mathbf{u}.$$



Stability analysis

For instance: inner product $y = \begin{bmatrix} a_1 & a_2 & a_3 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$:

$$
\begin{aligned}
\tilde{y} &= a_1 \otimes b_1 \oplus a_2 \otimes b_2 \oplus a_3 \otimes b_3 \\
&= a_1 b_1 (1+\delta_1) \oplus a_2 b_2 (1+\delta_2) \oplus a_3 b_3 (1+\delta_3) \\
&= \big(\big(a_1 b_1 (1+\delta_1) + a_2 b_2 (1+\delta_2)\big)(1+\delta_4) + a_3 b_3 (1+\delta_3)\big)(1+\delta_5) \\
&= a_1 b_1 + a_2 b_2 + a_3 b_3 + (\delta_1 + \delta_4 + \delta_5) a_1 b_1 + (\delta_2 + \delta_4 + \delta_5) a_2 b_2 \\
&\quad + (\delta_3 + \delta_5) a_3 b_3 + o(\mathbf{u})
\end{aligned}
$$

$\delta_1 \delta_4$

Taking absolute values + using $|\delta_i| \le \mathbf{u}$:

$$|\tilde{y} - y| \le 3\mathbf{u}(|a_1||b_1| + |a_2||b_2| + |a_3||b_3|) + o(\mathbf{u}).$$

$$= \begin{bmatrix} |a_1| & |a_2| & |a_3| \end{bmatrix} \begin{bmatrix} |b_1| \\ |b_2| \\ |b_3| \end{bmatrix}$$

Error in inner products

Error bounded by $|a|^T|b|$ (where $|v|$ is componentwise).
It may be a lot larger than $a^T b$, for instance in

$$\begin{bmatrix} 1 & -1 & 1 \end{bmatrix} \begin{bmatrix} 10^6+1 \\ 10^6 \\ 1 \end{bmatrix} = 1+1=2$$

Similarly, one can bound $|\widetilde{AB} - AB| \leq n|A||B|\mathbf{u} + o(\mathbf{u})$.

. . . but that's a lot of algebra, already for a simple problem.

$$\begin{bmatrix} 1 & |-1| & 1 \end{bmatrix} \begin{pmatrix} 10^6+1 \\ 10^6 \\ 1 \end{pmatrix} = 2\cdot 10^6 + 2$$

Backward stability

Trick (Wilkinson, $\approx$ 1960s) for more complicated problems: see $\tilde{y}$ as the exact output of running our algorithm on a perturbed input. For instance, in the problem above:

$$\begin{aligned}\tilde{y} &= \dots \\ &= ((a_1 b_1(1+\delta_1) + a_2 b_2(1+\delta_2))(1+\delta_4) + a_3 b_3(1+\delta_3))(1+\delta_5) \\ &= a_1\widetilde{b_1} + a_2\widetilde{b_2} + a_3\widetilde{b_3}\end{aligned}$$

with

$$\begin{aligned}\widetilde{b_1} &= b_1(1+\delta_1)(1+\delta_4)(1+\delta_5) = b_1 + 3\mathbf{u}b_1 + o(\mathbf{u}), \\ \widetilde{b_2} &= b_2(1+\delta_2)(1+\delta_4)(1+\delta_5) = b_2 + 3\mathbf{u}b_2 + o(\mathbf{u}), \\ \widetilde{b_3} &= b_3(1+\delta_2)(1+\delta_5) = b_3 + 2\mathbf{u}b_3 + o(\mathbf{u}).\end{aligned}$$

$|\tilde{b}-b| \leq 3u|b|$

$$\|\tilde{b}-b\| = \sqrt{(\tilde{b}_1-b_1)^2+(\tilde{b}_2-b_2)^2+(\tilde{b}_3-b_3)^2} :$$

$$= \sqrt{(3u\,b_1)^2+(3u\,b_2)^2+(3u\,b_3)^2} = \sqrt{(3u)^2(b_1^2+b_2^2+b_3^2)} =$$

$$= 3u\sqrt{b_1^2+b_2^2+b_3^2} = 3u\,\|b\|$$

$$y = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \begin{bmatrix} b_1 & b_2 & b_3 \end{bmatrix} = \begin{bmatrix} a_1 b_1 & a_1 b_2 & a_1 b_3 \\ a_2 b_1 & a_2 b_2 & a_2 b_3 \\ a_3 b_1 & a_3 b_2 & a_3 b_3 \end{bmatrix}$$

$$\tilde{y} = \begin{bmatrix} a_1 \otimes b_1 & \cdots \\ a_2 \otimes b_1 & \cdots \\ a_3 \otimes b_1 & \cdots \end{bmatrix} = \begin{bmatrix} a_1 b_1 (1+\delta_1) & a_1 b_2 (1+\delta_4) & a_1 b_3 (1+\delta_7) \\ a_2 b_1 (1+\delta_2) & a_2 b_2 (1+\delta_5) & a_2 b_3 (1+\delta_8) \\ a_3 b_1 (1+\delta_3) & a_3 b_2 (1+\delta_6) & a_3 b_3 (1+\delta_9) \end{bmatrix}$$

Backward stability

Hence

$$\frac{\|\tilde{y} - y\|}{\|y\|} \leq \kappa_{rel}(\text{inner product}, a, b) \frac{\|\widetilde{b} - b\|}{\|b\|}.$$

with $\frac{\|\widetilde{b}-b\|}{\|b\|} \leq 3\mathbf{u} + o(\mathbf{u})$.

TL;DR: apart from a factor 3, our algorithm is as accurate as it could get (given the unavoidable intrinsic error). Nothing catastrophic will happen.

Warning the 'see the error as modified input' trick does not work on all algorithms.

Backward stable algorithms are as accurate as theoretically possible (given the condition number of a problem), up to some factor that depends only on the dimension (e.g., n, $2n^2 + 18n$, ...).

Exercises

1. Show that solving a linear system $Tx = b$ with a triangular matrix T is backward stable, as a function of T. (Hint: expand errors as for the inner product example, and define a modified matrix $\widetilde{T}$).