Algorithm Stability and Big O / small o notation

Friday, September 6, 2024 12:40 PM

Class 05: September 6, 2024

Recall: We defined the concepts of condition number (for a ***mathematical problem***) and numerical stability (for an **algorithm**). These two have to do with the relationship between *known* **relative errors for the inputs** (or estimates) and the *expected, unknown* **relative error in the outputs**. To make this more concrete, we imagine that the mathematical problem can be written as a function **y = f(x)**, where x is our input and y is our output.

We defined the condition number of a function f at x, denoted as k_f(x), as the limit as a perturbation Delta x goes to zero of the relative error between f(x+Deltax) and f(x), and the relative error |Delta x| / |x|. If f(x) is a differentiable function at x, we can come up with the formula k_f(x) = |f'(x)| |x| / |f(x)|.

A math problem is said to be **"well conditioned"** if the condition number is close to 1: it does not amplify relative errors much. It is said to be **"ill conditioned"** if condition number is very big; if it is around 10^p, that means you lose about p digits of relative accuracy. Some problems in the sciences and engineering are such that small perturbations in inputs cause big differences in outputs (e.g. chaotic systems, inverse problems in design), and so we expect to *lose a lot of precision even if we evaluated f(x) with perfect accuracy*.

Stability is the corresponding property (in terms of loss of precision, or error propagation) for an algorithm trying to approximate f(x). A stable algorithm is one that will not lose precision needlessly: it will give us a relative error in outputs that is about the same as the condition number suggests. An unstable algorithm is one that can potentially lose quite a bit of precision, even for a well behaved problem.

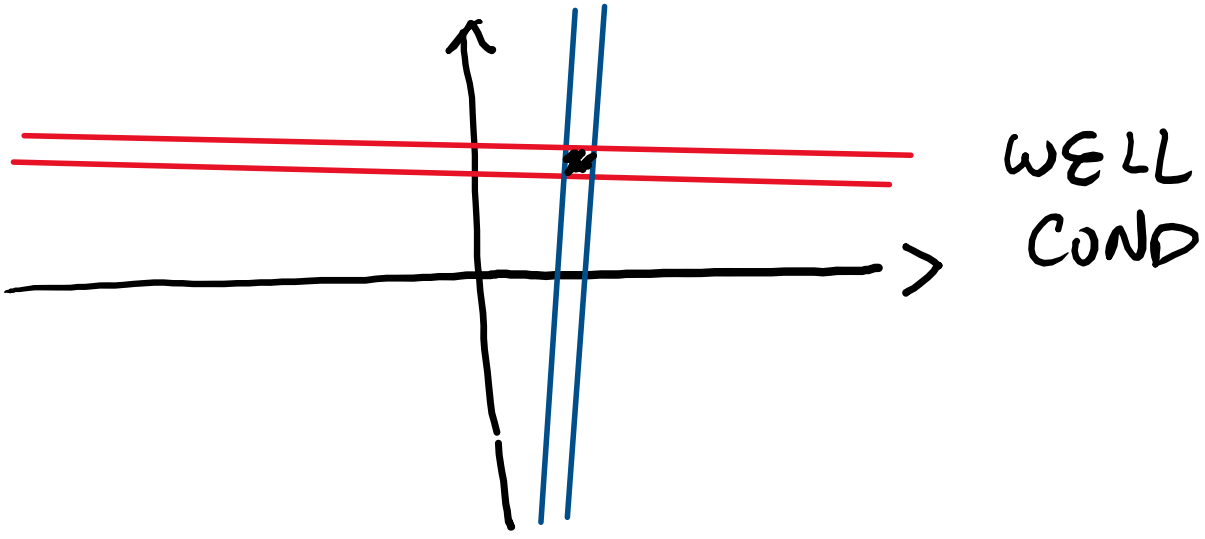
Finally, we went over a number of examples of unstable and stable algorithms applied to the same problem.

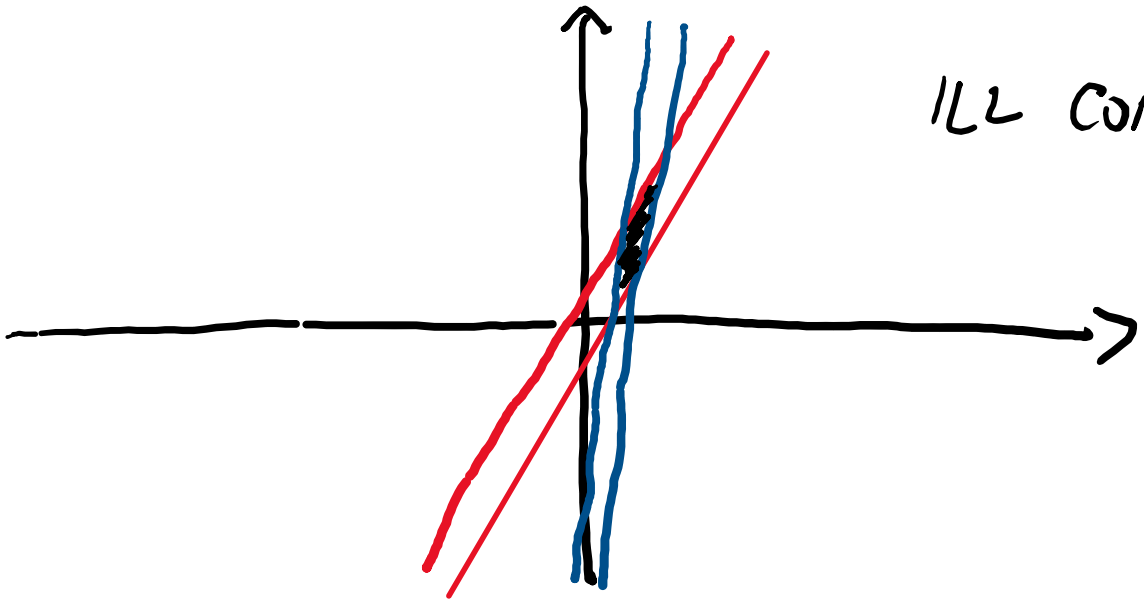
CODE DEMO : STABILITY

LINEAR SYSTEM

$$\begin{bmatrix} \varepsilon & 1 \\ 1 & -\varepsilon \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

INTUITION FROM GEOMETRY





The problem above is well cond, but depending on the order of elimination, our algorithm is unstable or stable!

"Big O and little o notation"

- Quantity q and our algorithm might involve approx $\tilde{q}(h)$,

$$\lim_{h \to 0} \tilde{q}(h) = q$$

$\sin h \approx h$ for small h.

How fast does $\tilde{q}(h) \to q$ and what can I say about $\varepsilon(h) = q(h) - q$?

$$\varepsilon(h) = 2h^2 + 3h^3 + 4h^4 + \dots$$

"$\varepsilon(h) = O(h^2)$"

$$\log(\varepsilon(h)) \approx \log(Ch^2)$$

$$\approx 2\log(h) + \log(C)$$

$$\frac{\varepsilon(h)}{h^2} = 2 + 3h + 4h^2 + \dots$$

"$q(h) = q + O(h^2)$"

DEF → Let $f(x)$, $g(x)$ defined around $x = 0$. We say that $f(x) = O(g(x))$ if as $x \to 0$ $\exists$ $M, \delta > 0$ s.t.

$$\left|\frac{f(x)}{g(x)}\right| \leq M \quad \text{if } |x| < \delta$$

In particular, if $\lim \left|\frac{f(x)}{g(x)}\right| = L$ exists,

In particular, if $\lim_{x \to 0} \left| \frac{f(x)}{g(x)} \right| = L$ exists, this is true.

Little o → "f goes faster than g"

$f(x) = o(g(x))$ $\quad \lim_{x \to 0} \frac{f(x)}{g(x)} = 0$

$f(x) = \sin x \qquad f_1(x) = x$

$f_3(x) = x - x^3/3!$

Maclaurin exp. f:

$$\sin x = x - x^3/3! + x^5/5! - \dots$$

Taylor remainder

$$\sin x = x + \underbrace{\frac{\sin(n)}{3!} x^3}$$

n between 0 and x.

$$\sin x = x - x^3/3! + \underbrace{\frac{\sin(\mu)}{5!} x^5}$$

Iterative Algorithms

To a sequence a_n

QoI q, Sequence q_n

$$\lim_{n\to\infty} q_n = q$$

$$q_n - q \approx 1/n^2 + 3/n^3 - 4/n^4 \cdots$$

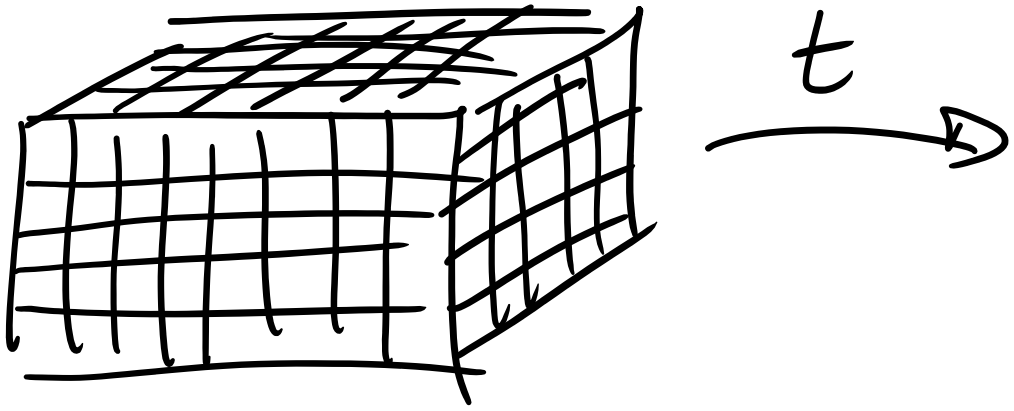
"$q_n - q$ is $O(1/n^2)$" as $n \to \infty$.

Big O for costs:

LINEAR SYSTEM $A\vec{x} = b$.

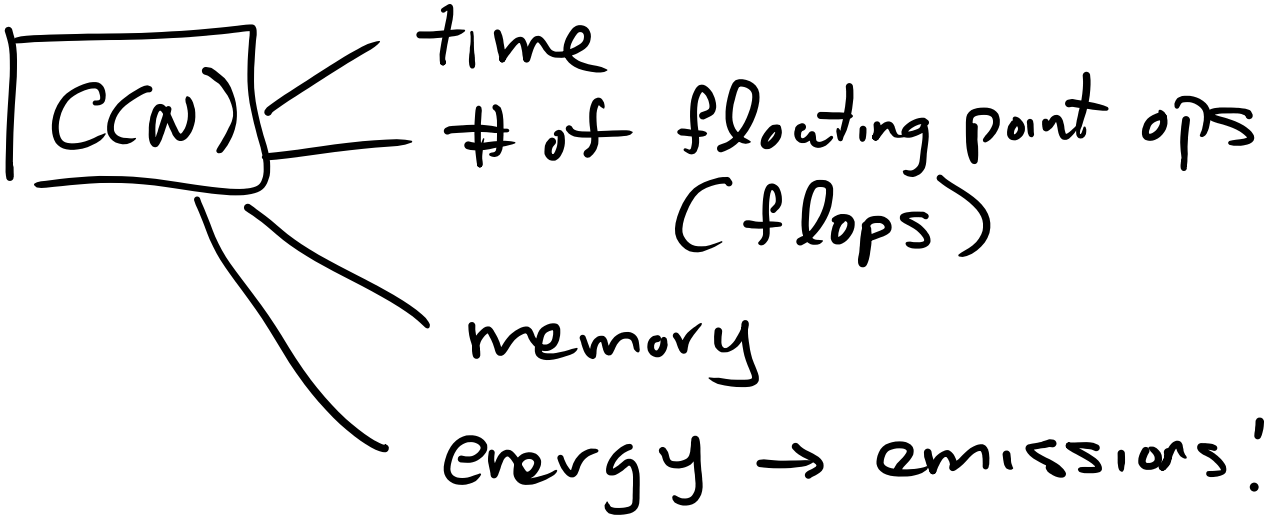
$N \times N$ $\quad N \times 1$

$N \to$ # variables



How does computational cost $C(N)$ grow as $N \to \infty$.

time



$$C(N) = \underline{N^3} + 2N^2 - N + 10$$

$$C(N) = O(N^3)$$

$$\lim_{N \to \infty} \frac{C(N)}{N^3} = 1$$