Ch 1: Stability

Wednesday, August 27, 2025 7:20 PM

Learning Objective:
- Distinguish "conditioning" from "stability" in our context

See Demos/Ch1_stability_simple.ipynb

... previously, we discussed whether a "problem" is well-conditioned

we abstracted this to $f(x)$ but it could be as complex as predicting the # of hurricanes next year (and x = current weather observations)

now, discuss stability of an algorithm.

To clarify, let our problem be evaluating $f(x) = (x-c)^2$

If this is ill-conditioned then no matter how clever we are at writing code, we may lose a lot of precision. So conditioning is a prerequisite for accurate computations

ie. small condition #

But, there are many ways to implement f, ie, different algorithms

Algorithm 1: temp $= x - c$
output $=$ temp^2

Algorithm 2: output $= x^2 - 2x \cdot c + c^2$

To summarize:

Problem (ie, overall goal) can be well-conditioned or ill-conditioned

Algorithms (i.e., implementations) can be stable or unstable

So what does Stable mean?

In some contexts (PDEs, linear operators) we have precise definitions

For now, use the book's vague definition

[An algorithm is stable if] small changes in the initial data produce correspondingly small changes in the final results.

i.e., errors don't compound over time

Another (vague) definition:

An algorithm is unstable if it produces more error than would be expected by the amount of ill-conditioning

i.e., if the relative condition # κ is 10^7, this means we expect to lose about 7 digits of precision.

If our algorithm loses 10 digits, it's an unstable algo.

Ch 1: Stability, p. 2

Thursday, August 28, 2025 8:13 AM

How do we know if an algorithm is stable or not?

1. Compute the baseline: what's the conditioning of the problem?
2. For the algorithm, compute the conditioning of every step (or at least suspicious ones), and if any of these significantly exceeds the conditioning of the problem, declare the algorithm unstable

Note: because of our chain rule property, product of conditioning across all steps is the same, but for algorithms, we assume error is introduced at all stages, so we don't allow "cancellations" of condition numbers across steps.

EXAMPLE: $f(x) = (x-c)^2 = x^2 - 2\cdot c\cdot x + c^2$

1. Compute $\kappa_f(x) := \left|\frac{x}{f(x)}\cdot f'(x)\right| = \left|\frac{x}{(x-c)^2}\cdot 2\cdot(x-c)\right| = 2\left|\frac{x}{x-c}\right|$ } BASELINE

Moderately ill-conditioned when $x \approx c$, otherwise well-conditioned

eg, $x = 3+10^{-5}$ then $\kappa_f(x) \approx 2\cdot 10^5$

2. Algo 1: $y = x-c$
$z = y^2$
return z

Step 1: $y(x) = x-c$
$\kappa_y(x) = \left|\frac{x}{y(x)}\cdot y'(x)\right| = \left|\frac{x}{x-c}\right|$

Not good, but no worse than baseline

Step 2: $z(y) = y^2$
$\kappa_z(y) = \left|\frac{y}{z(y)}\cdot z'(y)\right| = \left|\frac{y}{y^2}\cdot 2y\right| = 2$

Good.

Overall, no single step significantly exceeded our baseline, so Algo 1 is stable.

Algo 2: $y = -2\cdot c\cdot x$
$z = x^2 + y + c^2$
return z

Step 1: $y(x) = -2\cdot c\cdot x$
$\kappa_y(x) = \left|\frac{x}{y(x)}\cdot y'(x)\right| = \left|\frac{x}{2\cdot c\cdot x}\cdot 2\cdot c\right| = 1$

Good, no issues

Step 2: even constants, if they are floats
$z(x,y,c) = x^2 + y + c^2$

multivariate! what to do? Check for each input at a time, and if any of them are ill-conditioned, the algo is ill-conditioned

$\kappa_z(x) = \left|\frac{x}{z(x)}\cdot\frac{\partial z}{\partial x}\right| = \left|\frac{x\cdot 2x}{x^2+y^2+c^2}\right| = 2\frac{x^2}{(x-c)^2}$ plug in $y = -2cx$

Bad! We've squared the condition number... algo 2 is unstable

No need to bother checking $\kappa_z(y)$ or $\kappa_z(c)$

We "lose" twice as many digits (when $x \approx c$) as we need to.

Ch 1: Stability, p. 3

Thursday, August 28, 2025 8:13 AM

Supplemental
(Didn't cover in class)

Backward Error cf. Driscoll + Braun

A useful perspective

Suppose we want to find $y = f(x)$ ← still could represent something complicated (and often multi-dimensional)

but we do it with our imperfect algorithm $\tilde{f}$

(for now, assume perfect input x)

Then we make an error, i.e., if $\tilde{y} = \tilde{f}(x)$

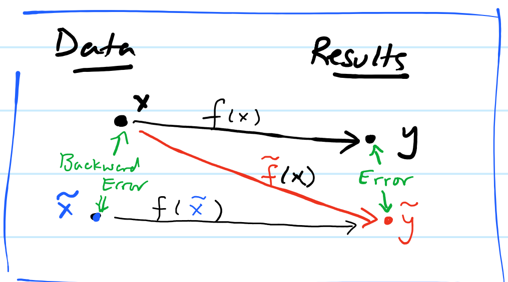
then $\tilde{y} \neq y$ (and $|\tilde{y} - y|$ is the "error")

[Recall, for conditioning, we assumed a perfect algorithm but imperfect input data ... so exactly the opposite scenario

But, suppose we can find $\tilde{x}$ such that $\tilde{y} = f(\tilde{x})$

i.e., we have the right answer ($\tilde{y}$) to the wrong question/problem ($\tilde{x}$)

Then we say $|\tilde{x} - x|$ is the "backward error"



Turns out, if an algorithm always produces small backwards error, then it is stable. (but not vice-versa)

Further Reading: ch 1 in Driscoll + Braun (SIAM 2018)