Lecture 14 - *Stability*

OBJECTIVE:
It would be nice if we could get exact solutions to numerical problems.
But, the reality is that because problems are continuous while computer arithmetic is discrete, this is not generally possible.
Stability tells us what is possible (or what we can expect) when solving a problem.
i.e., it tells us what it means to get the "right answer" even if this is not the exact answer.

◇ ALGORITHMS

In Lecture 12, we said that an abstract way to think of solving a problem is like evaluating a function

$$y = f(x)$$

where x represents the input to the problem (the data), f represents the "problem" itself, and y represents its solution.

An *algorithm* can be viewed as a different function $\tilde{f}$ that usually takes the same data (actually the rounded data) and maps it to a different solution $\tilde{f}(x)$.

e.g., The computer code used to implement the algorithm is viewed as the $\tilde{f}$. So, even if two different implementations are meant to produce the same result, these are generally two different functions $\tilde{f}_1$ and $\tilde{f}_2$.

◇ ACCURACY

A good algorithm should have an $\tilde{f}$ that closely approximates the underlying problem f.

If nothing else, $\tilde{f}$ will be affected by rounding error during its execution.

To quantify things, we may introduce the *absolute error* of a computation

$$\|\tilde{f}(x) - f(x)\|$$

or the *relative error*

$$\frac{\|\tilde{f}(x) - f(x)\|}{\|f(x)\|}$$

$\longrightarrow$ We will mostly consider relative errors.

If $\tilde{f}$ is a good algorithm, we might expect the relative error to be small ($\mathcal{O}(\epsilon_{\text{machine}})$).

We say that $\tilde{f}$ is an *accurate* algorithm for f if for all (relevant) input data x

$$\frac{\|\tilde{f}(x) - f(x)\|}{\|f(x)\|} = \mathcal{O}(\epsilon_{\text{machine}}) \tag{1}$$

We will clarify the meaning of $\mathcal{O}(\epsilon_{\text{machine}})$ shortly.

◇ STABILITY

If f is ill-conditioned, the goal of achieving (1) is unreasonable.
$\longrightarrow$ rounding of input is inevitable, so even if the algorithm could somehow do everything exactly from there, $\|\tilde{f}(x) - f(x)\|$ may still be large!

So instead of always aiming for accuracy, the most we can (always) reasonably aim for is *stability*:

We say that an algorithm $\tilde{f}$ for a problem f is *stable* if for all (relevant) input data x

$$\frac{\|\tilde{f}(x) - f(\overline{x})\|}{\|f(\overline{x})\|} = \mathcal{O}(\epsilon_{\text{machine}})$$

for some $\overline{x}$ satisfying

$$\frac{\|\overline{x} - x\|}{\|x\|} = \mathcal{O}(\epsilon_{\text{machine}})$$

In words, the best we can hope for is a stable algorithm; i.e., one that gives nearly the right answer to nearly the right question.

Note 1. $\mathcal{O}(\epsilon_{\text{machine}})$ *is too strict for other numerical problems such as solution of differential equations where there are many more "layers" of approximations made.*

◇ BACKWARD STABILITY

Many algorithms in numerical linear algebra satisfy a condition that is both simpler and stronger than our definition of stability.

We say an algorithm $\tilde{f}$ for a problem f is *backward stable* if for all (relevant) input data x

$$\tilde{f}(x) = f(\tilde{x})$$

for some $\tilde{x}$ satisfying

$$\frac{\|\tilde{x} - x\|}{\|x\|} = \mathcal{O}(\epsilon_{\text{machine}})$$

Note 2. *This is stronger than the definition of stability (why?).*

In words, a backward stable algorithm gives *exactly* the right answer to nearly the right question.

$\diamond$ $\mathcal{O}(\epsilon_{\text{machine}})$

We use the concept of

$$\| \text{ computed quantity } \| = \mathcal{O}(\epsilon_{\text{machine}})$$

in a sense that has a few assumptions built into it:

- $\|$ computed quantity $\|$ means the norm of some number(s) computed by some algorithm $\tilde{f}$ for a problem f, depending on both in the input data x for f and on $\epsilon_{\text{machine}}$.
  e.g., relative error.

Note 3. *Provided the input and output are finite-dimensional (always the case for this course) the norm used is not relevant.*

Theorem 1. *For finite-dimensional inputs and outputs, the properties of accuracy, stability, and backward stability all hold or fail to hold independently of the choice of norm.*

Note 4. *The only effect from one choice of norm to another is the constant buried in the $\mathcal{O}(\epsilon_{\text{machine}})$ notation.*

- There is an implicit process of $\epsilon_{\text{machine}} \to 0$.

Of course, this is nonsense for a fixed floating-point system.

One should imagine instead a series of computations done in higher and higher precision (perhaps on different computers!)
e.g. single precision, double precision, quadruple precision, etc...

Then $\|\,\text{computed quantity}\,\| \to 0$ as precision is increased.

- $\mathcal{O}(\cdot)$ applies *uniformly* to all data x.

i.e., the constant buried in the $\mathcal{O}(\epsilon_{\text{machine}})$ notation can be specified *independently of* x (it does not depend on x).

◇ DEPENDENCE ON m AND n, NOT $\mathbf{A}$ AND $\mathbf{b}$

The uniformity of the constant implicit in the $\mathcal{O}(\epsilon_{\text{machine}})$ notation can be illustrated by the following example.

Suppose we are considering an algorithm for solving $\mathbf{Ax} = \mathbf{b}$ where $\mathbf{A} \in \mathbb{R}^{m \times m}$ is nonsingular.

Suppose the computed result $\tilde{x}$ for this algorithm satisfies

$$\frac{\|\tilde{\mathbf{x}} - \mathbf{x}\|}{\|\mathbf{x}\|} = \mathcal{O}(\kappa(\mathbf{A})\epsilon_{\text{machine}})$$

i.e.,

$$\frac{\|\tilde{\mathbf{x}} - \mathbf{x}\|}{\|\mathbf{x}\|} \leq C\kappa(\mathbf{A})\epsilon_{\text{machine}}$$

for some constant C that does not depend on $\mathbf{A}$ or $\mathbf{b}$ and $\epsilon_{\text{machine}}$ sufficiently small.

If $\|\mathbf{x}\| = 0$, then this should be understood to be

$$\|\tilde{\mathbf{x}} - \mathbf{x}\| \leq C\kappa(\mathbf{A})\epsilon_{\text{machine}}\|\mathbf{x}\|$$

i.e., $\tilde{\mathbf{x}} = \mathbf{x}$ when $\epsilon_{\text{machine}}$ is sufficiently small.

However, although C does not depend on $\mathbf{A}$ or $\mathbf{b}$, it usually depends on m!

Strictly speaking, as m changes, so does the problem.

In practice, one would expect the effects of rounding error to grow as m, n increase (there are simply more floating-point operations!)

So, in principle C contains a rapidly growing dependence on m or n that would make such a bound useless in practice.

However, this is a bit of worst-case analysis and is rarely seen in practice due to statistical (random) nature round-off errors
$\rightarrow$ there tends to be at least some cancellation of round-off errors (although Gaussian elimination is the only exception we will discuss at some point).

Hence, in a real calculation on a real machine, $\mathcal{O}(\epsilon_{\mathrm{machine}})$ will be assumed to have buried in it a constant of at most 100 or 1000.