# Lecture 15 - *More on Stability*

**OBJECTIVE:** We now consider a few examples of stable and unstable algorithms.
Then, we proceed to discuss the powerful concept of backward error analysis which links conditioning and stability.

## $\diamond$ STABILITY OF FLOATING-POINT ARITHMETIC

The four simplest computational problems (functions) are $+, -, \times, /$.

In this case, we do not go into algorithmic details!

Naturally, we use the floating-point analogues in our analysis: $\oplus, \ominus, \otimes, \oslash$ It turns out that the axioms

$$\begin{aligned} \mathrm{fl}(x) &= x(1+\epsilon) \qquad |\epsilon| \le \epsilon_{\text{machine}} \\ x \circledast y &= (x * y)(1+\epsilon) \end{aligned}$$

imply that these most basic arithmetic operations are backward stable.

Let us show this for $\ominus$ since one might suspect this has the greatest risk of instability.

input: $\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \in \mathbb{R}^2$

output: $x_1 - x_2 \in \mathbb{R}$

In functional form

$$f(x_1, x_2) = x_1 - x_2$$

So, our algorithm is

$$\tilde{f}(x_1, x_2) = \mathrm{fl}(x_1) \ominus \mathrm{fl}(x_2)$$

i.e., first round $x_1, x_2$ to their nearest floating-point numbers, then apply floating-point subtraction.

Now,

$$\begin{aligned} \mathrm{fl}(x_1) &= x_1(1+\epsilon_1) \\ \mathrm{fl}(x_2) &= x_2(1+\epsilon_2) \end{aligned}$$

for some $\epsilon_1, \epsilon_2$ satisfying

$$|\epsilon_1|, |\epsilon_2| \leq \epsilon_{\text{machine}}$$

Also,

$$\mathrm{fl}(x_1) \ominus \mathrm{fl}(x_2) = [\mathrm{fl}(x_1) - \mathrm{fl}(x_2)]\,(1+\epsilon_3)$$

for some $\epsilon_3$ satisfying

$$|\epsilon_3| \leq \epsilon_{\text{machine}}.$$

Thus,

$$\begin{aligned}\mathrm{fl}(x_1) \ominus \mathrm{fl}(x_2) &= [x_1(1+\epsilon_1) - x_2(1+\epsilon_2)]\,(1+\epsilon_3) \\ &= x_1(1+\epsilon_1)(1+\epsilon_3) - \\ &\qquad x_2(1+\epsilon_1)(1+\epsilon_3) \\ &= x_1(1+\epsilon_4) - x_2(1+\epsilon_5)\end{aligned}$$

for some $\epsilon_4, \epsilon_5$ satisfying

$$|\epsilon_4|, |\epsilon_5| \leq 2\epsilon_{\text{machine}} + \mathcal{O}(\epsilon_{\text{machine}}^2) \qquad \text{(verify!)}$$

i.e.,

$$\tilde{f}(x) = \mathrm{fl}(x_1) \ominus \mathrm{fl}(x_2) = \tilde{x}_1 - \tilde{x}_2$$

where

$$\frac{|\tilde{x}_1 - x_1|}{|x_1|},\ \frac{|\tilde{x}_2 - x_2|}{|x_2|} = \mathcal{O}(\epsilon_{\text{machine}})$$

and any $c > 2$. Any norm on $\mathbb{R}^2$ now implies $\tilde{f}(x) = f(\tilde{x})$. i.e., floating-point subtraction is backward stable.

## $\diamond$ MORE EXAMPLES

**Example 15.1** INNER PRODUCT
It can be shown that the inner product of two vectors is backward stable (assignment).

**Example 15.2** OUTER PRODUCT
Given vectors $\mathbf{x} \in \mathbb{R}^m, \mathbf{y} \in \mathbb{R}^n$, compute the rank-one outer product $\mathbf{A} = \mathbf{x}\mathbf{y}^T$.

Obvious algorithm:
Set

$$\tilde{a}_{ij} = \mathrm{fl}(x_i) \otimes \mathrm{fl}(y_j)$$

$i = 1, 2, \ldots, m$
$j = 1, 2, \ldots, n$

This algorithm is stable, but not backwards stable (this is okay, by the way).

The reason it is not backward stable is because $\tilde{\mathbf{A}}$ will likely <u>not</u> have rank 1 $(\mathrm{rank}(\tilde{\mathbf{A}}) > 1)$.

So, it cannot be written as

$$\tilde{x}\tilde{y} = (x + \delta x)(y + \delta y)^T$$

In general when the dimension of the output exceeds that of the input, (in this case $mn > m+n$) algorithms are rarely backward stable.

**Example 15.3** ADDING 1
Let $x \in \mathbb{R}$ and suppose $f(x) = x + 1$.

Then, $\tilde{f}(x) = \mathrm{fl}(x) \oplus 1$

Again, this algorithm is stable, but not backward stable. It fails backward stability for $x \approx 0$.

In this case,

$$\begin{aligned}
\tilde{f}(x) &= \mathrm{fl}(x(1+\epsilon_1)+1) \\
&= 1 + \underbrace{x + x\epsilon_1 + \epsilon_2 + \epsilon_2 x} + \epsilon_1\epsilon_2 x \\
&= 1 + x(1 + \underbrace{\epsilon_1 + \epsilon_2 + \frac{\epsilon_2}{x}})
\end{aligned}$$

Can this be written as $\epsilon_3 = \mathcal{O}(\epsilon_{\text{machine}})$?

No! Not as $x \to 0$

So, $\tilde{x}$ cannot be expressed as $x + \delta x$ where $\delta x = \mathcal{O}(\epsilon_{\text{machine}})$.

i.e., Backward stability is a very special property and is only a reasonable goal for some problems but not others.

**Note 1.** *The problem to compute $x + y$ for data $x, y$ <u>is</u> backward stable.*

**Example 15.4**
It is only reasonable to expect stability and not backward stability when computing $\sin(x)$ or $\cos(x)$.

They both fail backward stability because there are points where the derivatives of $\sin(x)$ and $\cos(x)$ are zero.

e.g., Let $f(x) = \sin(x)$ for $x = \frac{\pi}{2} - \delta$ where $0 < \delta \ll 1$. Let's suppose we are lucky enough to get

$$\tilde{f}(x) = \mathrm{fl}(\sin(x))$$

i.e., the exact answer rounded to the floating-point system.

Now,

$$f'(x) = \cos(x) = \cos\left(\frac{\pi}{2} - \delta\right) = \sin(\delta) \approx \delta$$

So,

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

for some $\tilde{x}$ satisfying

$$|\tilde{x} - x| \approx \left|\frac{\tilde{f}(x) - f(\tilde{x})}{f'(x)}\right| \approx \frac{\mathcal{O}(\epsilon_{\text{machine}})}{\delta}$$

But since $\delta$ can be arbitrarily small,

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

$\diamond$ AN UNSTABLE ALGORITHM: Calculation of eigenvalues of a matrix by finding the roots of the characteristic polynomial

Since $\lambda$ is an eigenvalue of $\mathbf{A}$ if and only if $p(\lambda) = 0$, where $p$ is the characteristic polynomial det$(\mathbf{A} - \lambda\mathbf{I})$, the roots $\lambda$ of $p(\lambda) = 0$ are the eigenvalues of $\mathbf{A}$ (see Lecture 24).

This suggests the following algorithm:

1. Find the coefficients of the characteristic polynomial.

2. Find its roots.

This algorithm is *unstable* and SHOULD NEVER BE USED!

$\rightarrow$ Even when the problem is well-conditioned, this algorithm may produce answers with relative errors very much greater than $\epsilon_{\mathrm{machine}}$.

The instability comes from the root finding in Step 2.

Recall Example 12.5 (in the book), where we saw that the problem of finding the roots of a polynomial given its coefficients is generally *ill-conditioned*.

i.e., Small errors in the polynomial coefficients will be amplified when finding its roots - even if the root finding is done with perfect accuracy.

For example, let $\mathbf{A} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

The eigenvalues of $\mathbf{A}$ should be computed with errors $\mathcal{O}(\epsilon_{\text{machine}})$.

However, the above algorithm produces errors $\mathcal{O}(\sqrt{\epsilon_{\text{machine}}})$.

So, if $\epsilon_{\text{machine}} = 10^{-16}$, you lose half the digits: $\mathcal{O}(\sqrt{\epsilon_{\text{machine}}}) = 10^{-8}$

Try it on $\quad \mathbf{A} = \begin{bmatrix} 1 + 10^{-14} & 0 \\ 0 & 1 \end{bmatrix}$

(We don't use the identity matrix because the entries are too simple and under-represent the general case)

## $\diamond$ ACCURACY OF A BACKWARD STABLE ALGORITHM

Will a backward stable algorithm give accurate results?

The answer depends on the condition number $\kappa(x)$ of the problem
$\longrightarrow$ if $\kappa(x)$ is small, the relative errors will be small
$\longrightarrow$ if not, the accuracy suffers proportionately!

**Theorem 1.** *Suppose a backward stable algorithm is applied to solve a problem $f$ with condition number $\kappa$. Then, the relative errors satisfy*

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

<u>Proof</u>: By the definition of backward stability, $\tilde{f}(x) = f(\tilde{x})$ for some (relevant) data $x$ satisfying

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

By definition

$$\begin{aligned}
\kappa(x) &\geq \frac{\|f(\tilde{x}) - f(x)\|}{\|f(x)\|} \Bigg/ \frac{\|\tilde{x} - x\|}{\|x\|} \\
&= \frac{\|\tilde{f}(x) - f(x)\|}{\|f(x)\|} \Bigg/ \frac{\|\tilde{x} - x\|}{\|x\|}
\end{aligned}$$

we have

$$\begin{aligned}
\frac{\|\tilde{f}(x) - f(x)\|}{\|f(x)\|} &\leq \frac{\|\tilde{x} - x\|}{\|x\|}\kappa(x) \\
&= \mathcal{O}(\kappa(x)\epsilon_{\text{machine}})
\end{aligned}$$

## $\diamond$ BACKWARD ERROR ANALYSIS

The process of obtaining an accuracy estimate by two steps

1. investigate the conditioning of the problem

2. investigate the stability of the algorithm

is called *backward error analysis*.

$\rightarrow$ If the algorithm is stable, the accuracy reflects the condition number.

This is not the most obvious way to do error analysis!

A more intuitive approach is *forward error analysis*, where careful account of all rounding errors is kept as an algorithm is executed.

Experience has shown that it is harder to carry out this analysis for the algorithms of numerical linear algebra.

With the aid of hindsight, one can see this as follows:

Suppose you had a reliable method for solving $\mathbf{A}\mathbf{x} = \mathbf{b}$.

It is a fact that the results are consistently less accurate when $\mathbf{A}$ is ill-conditioned.

To ask forward analysis to extract the concept of condition number from individual flops is a pretty tall order!

But, it would have to somehow, in order to produce the correct answer.

BOTTOM LINE: It is not possible to do better than to compute the exact solution to a perturbed problem. Backward error analysis is a reflection of this backward reality.