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 how we perform the computation.

Sensitivity/conditioning: tells you if you have a bad problem.
Stability: tells you if you are using a bad algorithm to solve it.

Floating point numbers in a nutshell

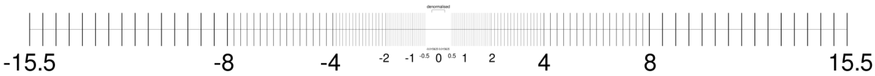
TL;DR Floating point numbers are numbers in base-2 scientific (exponential) notation.

`double` (64-bit numbers):

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

Plus special numbers like like `0`, `-0`, `Inf` and `NaN`.

Smaller numbers are packed more densely:



Representation error

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...

There are non-representable numbers, even simple ones such as $\frac{1}{10} = 0.1_{\text{dec}} = 0.0\overline{0011}_{\text{bin}}$.

Storing numbers required approximations and rounding. This can lead to unexpected inexactness, exactly as in (decimal) $\frac{1}{3} + \frac{1}{3} + \frac{1}{3} = 0.33333 + 0.33333 + 0.33333 = 0.99999 \neq 1$.

Rounding error bound

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

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, if you have available `double f(double x)`!
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:

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

The intrinsic error in a computation (due to inaccuracy in input) is $\kappa_{rel}(f, x)\mathrm{u}$.

Stability analysis

Apart from lucky cases (e.g., when all inputs and intermediate results are exactly representable, or when errors cancel out), you can't expect to compute $y = f(x)$ with better (relative) error than $\kappa_{rel}(f, x)\mathrm{u}$.

High $\kappa_{rel} \implies$ bad problem: no algorithm can compute the result accurately.

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

Definition

An algorithm is called stable if it computes its output up to an error of the same order of magnitude of the intrinsic error $\kappa_{rel}(f, \mathbf{x})\mathrm{u}$.

An algorithm can be stable on some inputs, and unstable on others.

Stability: a priori and a posteriori

Proving stability directly requires a lot of tedious computations to keep track of the errors.

Rounding appears in two places: (1) the inputs, (2) the result of each operation. We have already assessed the impact of (1), in first-order.

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

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

Proving stability

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

$$\begin{aligned} \tilde{y} &= a_1 \otimes b_1 \oplus a_2 \otimes b_2 \oplus a_3 \otimes b_3 \\ &= a_1 b_1 (1+\delta_1) \oplus a_2 b_2 (1+\delta_2) \oplus a_3 b_3 (1+\delta_3) \\ &= ((a_1 b_1 (1+\delta_1) + a_2 b_2 (1+\delta_2))(1+\delta_4) + a_3 b_3 (1+\delta_3))(1+\delta_5) \\ &= a_1 b_1 + a_2 b_2 + a_3 b_3 + (\delta_1 + \delta_4 + \delta_5) a_1 b_1 + (\delta_2 + \delta_4 + \delta_5) a_2 b_2 \\ &\quad + (\delta_3 + \delta_5) a_3 b_3 + (\text{terms with products of two or more } \delta_i\text{'s}) \end{aligned}$$

Taking absolute values and using $|\delta_i| \leq \mathrm{u}$:

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

Error in inner products

Theorem

If $y = \mathbf{a}^T\mathbf{b}$, then

$$|\tilde{y} - y| \leq 3\mathrm{u}|\mathbf{a}|^T|\mathbf{b}|$$

(componentwise absolute value).

It may be a lot larger than $\mathbf{a}^T\mathbf{b}$, for instance in

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

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

Also, this conflates the two issues: is this error high because of a bad problem, or because of a bad algorithm?

Backward stability

Trick (Wilkinson, $\approx$ 1960s): sometimes we can see $\tilde{y}$ as the exact output of running our algorithm on a perturbed input. For instance, above:

$$\begin{aligned}\tilde{y} &= \dots \\ &= ((a_1 b_1(1+\delta_1) + a_2 b_2(1+\delta_2))(1+\delta_4) + a_3 b_3(1+\delta_3))(1+\delta_5) \\ &= a_1 \hat{b}_1 + a_2 \hat{b}_2 + a_3 \hat{b}_3\end{aligned}$$

with

$$\begin{aligned}\hat{b}_1 &= b_1(1+\delta_1)(1+\delta_4)(1+\delta_5), \\ \hat{b}_2 &= b_2(1+\delta_2)(1+\delta_4)(1+\delta_5), \\ \hat{b}_3 &= b_3(1+\delta_2)(1+\delta_5).\end{aligned}$$

For each $i = 1, 2, 3$ we have $\quad \dfrac{|\hat{b}_i - b_i|}{|b_i|} \leq 3\mathrm{u} + o(\mathrm{u})$

Backward stability

Exact and inexact

$$\tilde{y}(\mathbf{a}, \mathbf{b}) = y(\mathbf{a}, \hat{\mathbf{b}}).$$

Hence

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

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

Apart from a factor equal to the dimension $n = 3$, our algorithm is as accurate as it could get (given the unavoidable intrinsic error).

Backward stability: definition

Definition

An algorithm to compute $\mathbf{y} = f(\mathbf{x})$ is called backward stable if the computed output $\tilde{\mathbf{y}}$ can be written as $\tilde{\mathbf{y}} = f(\hat{\mathbf{x}})$, where
$$\frac{\|\hat{\mathbf{x}} - \mathbf{x}\|}{\|\mathbf{x}\|} = O(\mathrm{u}).$$
In real-life usage, this $O(\mathrm{u})$ notation often hides polynomial factors in the dimension n: e.g., $n\mathrm{u}$, $(2n^2 + 18n)\mathrm{u}$, ...

Backward stable algorithms are as accurate as theoretically possible (given the condition number of a problem), up to that big-O.

Proof:
$$\frac{\|\tilde{\mathbf{y}} - \mathbf{y}\|}{\|\mathbf{y}\|} \leq \kappa_{rel}(f, \mathbf{x}) \frac{\|\hat{\mathbf{x}} - \mathbf{x}\|}{\|\mathbf{x}\|} = \kappa_{rel}(f, \mathbf{x}) O(\mathrm{u}),$$

while the best attainable accuracy is $\kappa_{rel}(f, \mathbf{x})\mathrm{u}$.

A non-backward-stable algorithm

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

Example

Consider the problem of computing $f(\mathbf{a}, \mathbf{b}) = \mathbf{a}\mathbf{b}^T$ (rank-1 matrix) (with the obvious algorithm).
If the products $a_i b_j$ are performed approximately, the resulting columns are not all multiples of the same vector $\implies$ the result is not a rank-1 matrix $\widetilde{\mathbf{a}}\widetilde{\mathbf{b}}^T$.

Example (with exaggerated errors):

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \odot \begin{bmatrix} 4 & 5 & 6 \end{bmatrix} = \begin{bmatrix} 4.01 & 4.99 & 6.01 \\ 7.99 & 10.01 & 12.02 \\ 11.98 & 15.02 & 17.97 \end{bmatrix}$$

is not a rank-1 matrix $\hat{\mathbf{a}}\hat{\mathbf{b}}^T$.

Exercises

1. Show that the back-substitution algorithm $\mathbf{x} = f(T, \mathbf{y})$ to solve a linear system $T\mathbf{x} = \mathbf{y}$ with upper triangular T is backward stable, i.e., the computed $\tilde{\mathbf{x}}$ satisfies $\tilde{\mathbf{x}} = f(\hat{T}, \mathbf{y})$. (Hint: expand errors as for the inner product example, and define a modified matrix $\hat{T}$).

Book references: Trefethen–Bau, Lectures 13–15.