Numerical stability

From Wikipedia, the free encyclopedia

In the mathematical subfield of numerical analysis, **numerical stability** is a generally desirable property of numerical algorithms. The precise definition of stability depends on the context. One is numerical linear algebra and the other is algorithms for solving ordinary and partial differential equations by discrete approximation.

In numerical linear algebra, the principal concern is instabilities caused by proximity to singularities of various kinds, such as very small or nearly colliding eigenvalues. On the other hand, in numerical algorithms for differential equations the concern is the growth of round-off errors and/or small fluctuations in initial data which might cause a large deviation of final answer from the exact solution.[citation needed]

Some numerical algorithms may damp out the small fluctuations (errors) in the input data; others might magnify such errors. Calculations that can be proven not to magnify approximation errors are called *numerically stable*. One of the common tasks of numerical analysis is to try to select algorithms which are *robust* – that is to say, do not produce a wildly different result for very small change in the input data.

An opposite phenomenon is **instability**. Typically, an algorithm involves an approximative method, and in some cases one could prove that the algorithm would approach the right solution in some limit (when using actual real numbers, not floating-point numbers). Even in this case, there is no guarantee that it would converge to the correct solution, because the floating-point round-off or truncation errors can be magnified, instead of damped, causing the deviation from the exact solution to grow exponentially.[1]

Stability in numerical linear algebra [edit]

There are different ways to formalize the concept of stability. The following definitions of forward, backward, and mixed stability are often used in numerical linear algebra.

Consider the problem to be solved by the numerical algorithm as a function f mapping the data x to the solution y. The result of the algorithm, say y^*, will usually deviate from the "true" solution y. The main causes of error are round-off error and truncation error. The *forward error* of the algorithm is the difference between the result and the solution; in this case, $\Delta y = y^* - y$. The *backward error* is the smallest Δx such that $f(x + \Delta x) = y^*$; in other words, the backward error tells us what problem the algorithm actually solved. The forward and backward error are related by the condition number: the forward error is at most as big in magnitude as the condition number multiplied by the magnitude of the backward error.

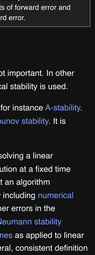


Diagram showing the **forward error** Δy and the **backward error** Δx, and their relation to the exact solution map f and the numerical solution f^*.

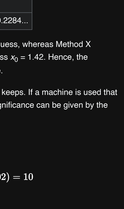
In many cases, it is more natural to consider the relative error

$$\frac{|\Delta x|}{|x|}$$

instead of the absolute error Δx.

The algorithm is said to be *backward stable* if the backward error is small for all inputs x. Of course, "small" is a relative term and its definition will depend on the context. Often, we want the error to be of the same order as, or perhaps only a few orders of magnitude bigger than, the unit round-off.

The usual definition of numerical stability uses a more general concept, called *mixed stability*, which combines the forward error and the backward error. An algorithm is stable in this sense if it solves a nearby problem approximately, i.e., if there exists a Δx such that both Δx is small and $f(x + \Delta x) - y^*$ is small. Hence, a backward stable algorithm is always stable.



Mixed stability combines the concepts of forward error and backward error.

An algorithm is *forward stable* if its forward error divided by the condition number of the problem is small. This means that an algorithm is forward stable if it has a forward error of magnitude similar to some backward stable algorithm.

Stability in numerical differential equations [edit]

The above definitions are particularly relevant in situations where truncation errors are not important. In other contexts, for instance when solving differential equations, a different definition of numerical stability is used.

In numerical ordinary differential equations, various concepts of numerical stability exist, for instance A-stability. They are related to some concept of stability in the dynamical systems sense, often Lyapunov stability. It is important to use a stable method when solving a stiff equation.

Yet another definition is used in numerical partial differential equations. An algorithm for solving a linear evolutionary partial differential equation is stable if the total variation of the numerical solution at a fixed time remains bounded as the step size goes to zero. The Lax equivalence theorem states that an algorithm converges if it is consistent and stable (in this sense). Stability is sometimes achieved by including numerical diffusion. Numerical diffusion is a mathematical term which ensures that roundoff and other errors in the calculation get spread out and do not add up to cause the calculation to "blow up". Von Neumann stability analysis is a commonly used procedure for the stability analysis of finite difference schemes as applied to linear partial differential equations. These results do not hold for nonlinear PDEs, where a general, consistent definition of stability is complicated by many properties absent in linear equations.

Example [edit]

Computing the square root of 2 (which is roughly 1.41421) is a well-posed problem. Many algorithms solve this problem by starting with an initial approximation x_0 to $\sqrt{2}$, for instance $x_0 = 1.4$, and then computing improved guesses x_1, x_2, etc. One such method is the famous Babylonian method, which is given by $x_{k+1} = (x_k + 2/x_k)/2$. Another method, called "method X", is given by $x_{k+1} = (x_k^2 - 2)^2 + x_k$.[note 1] A few iterations of each scheme are calculated in table form below, with initial guesses $x_0 = 1.4$ and $x_0 = 1.42$.

Babylonian	Babylonian	Method X	Method X
$x_0 = 1.4$	$x_0 = 1.42$	$x_0 = 1.4$	$x_0 = 1.42$
$x_1 = 1.4142857...$	$x_1 = 1.41422535...$	$x_1 = 1.4016$	$x_1 = 1.42026896$
$x_2 = 1.4142135564...$	$x_2 = 1.41421356242...$	$x_2 = 1.4028614...$	$x_2 = 1.42056...$
		...	...
		$x_{1000000} = 1.41421...$	$x_{27} = 7280.2284...$

Observe that the Babylonian method converges quickly regardless of the initial guess, whereas Method X converges extremely slowly with initial guess $x_0 = 1.4$ and diverges for initial guess $x_0 = 1.42$. Hence, the Babylonian method is numerically stable, while Method X is numerically unstable.

Numerical stability is affected by the number of the significant digits the machine keeps. If a machine is used that keeps only the four most significant decimal digits, a good example on loss of significance can be given by the two equivalent functions

$$f(x) = x\left(\sqrt{x+1} - \sqrt{x}\right) \text{ and } g(x) = \frac{x}{\sqrt{x+1} + \sqrt{x}}.$$

Comparing the results of

$$f(500) = 500\left(\sqrt{501} - \sqrt{500}\right) = 500\,(22.38 - 22.36) = 500(0.02) = 10$$

and

$$\begin{aligned} g(500) &= \frac{500}{\sqrt{501} + \sqrt{500}} \\ &= \frac{500}{22.38 + 22.36} \\ &= \frac{500}{44.74} = 11.17 \end{aligned}$$

by comparing the two results above, it is clear that loss of significance (caused here by catastrophic cancellation from subtracting approximations to the nearby numbers $\sqrt{501}$ and $\sqrt{500}$, despite the subtraction being computed exactly) has a huge effect on the results, even though both functions are equivalent, as shown below

$$\begin{aligned} f(x) &= x\left(\sqrt{x+1} - \sqrt{x}\right) \\ &= x\left(\sqrt{x+1} - \sqrt{x}\right)\frac{\sqrt{x+1} + \sqrt{x}}{\sqrt{x+1} + \sqrt{x}} \\ &= x\frac{\left(\sqrt{x+1}\right)^2 - \left(\sqrt{x}\right)^2}{\sqrt{x+1} + \sqrt{x}} \\ &= x\frac{x+1-x}{\sqrt{x+1} + \sqrt{x}} \\ &= x\frac{1}{\sqrt{x+1} + \sqrt{x}} \\ &= \frac{x}{\sqrt{x+1} + \sqrt{x}} \\ &= g(x) \end{aligned}$$

The desired value, computed using infinite precision, is 11.174755...[note 2]

See also [edit]

- Algorithms for calculating variance
- Stability theory
- Chaos theory
- Propagation of uncertainty

Notes [edit]

1. ^ This is a fixed point iteration for the equation $x = (x^2 - 2)^2 + x = f(x)$, whose solutions include $\sqrt{2}$. The iterates always move to the right since $f(x) \geq x$. Hence $x_1 = 1.4 < \sqrt{2}$ converges and $x_1 = 1.42 > \sqrt{2}$ diverges.
2. ^ The example is a modification of one taken from Mathews & Fink (1999).[2]

References [edit]

1. ^ A Giesela Engeln-Müllges; Frank Uhlig (2 July 1996). *Numerical Algorithms with C*. M. Schon (Translator), F. Uhlig (Translator) (1 ed.). Springer. p. 10. ISBN 978-3-540-60530-0.
2. ^ Mathews, John H.; Fink, Kurtis D. (1999). "Example 1.17". *Numerical Methods Using MATLAB* (3rd ed.). Prentice Hall. p. 28.

- Nicholas J. Higham (1996). *Accuracy and Stability of Numerical Algorithms*. Philadelphia: Society of Industrial and Applied Mathematics. ISBN 0-89871-355-2.
- Richard L. Burden; J. Douglas Faires (2005). *Numerical Analysis* (8th ed.). U.S.: Thomson Brooks/Cole. ISBN 0-534-39200-8.
- Mesnard, Olivier; Barba, Lorena A. (2017). "Reproducible and Replicable Computational Fluid Dynamics: It's Harder Than You Think". *Computing in Science & Engineering*. **19** (4): 44–55. arXiv:1605.04339. Bibcode:2017CSE....19d..44M. doi:10.1109/MCSE.2017.3151254. S2CID 11288122.

v • t • e **Linear algebra** [hide]
Outline • Glossary
Basic concepts: Scalar • Vector • Vector space • Scalar multiplication • Vector projection • Linear span • Linear map • Linear projection • Linear independence • Linear combination • Multilinear map • Basis • Change of basis • Row and column vectors • Row and column spaces • Kernel • Eigenvalues and eigenvectors • Transpose • Linear equations
Matrices: Block • Decomposition • Invertible • Minor • Multiplication • Rank • Transformation • Cramer's rule • Gaussian elimination
Bilinear: Orthogonality • Dot product • Hadamard product • Inner product space • Outer product • Kronecker product • Gram–Schmidt process
Multilinear algebra: Determinant • Cross product • Triple product • Seven-dimensional cross product • Geometric algebra • Exterior algebra • Bivector • Multivector • Tensor • Outermorphism
Vector space constructions: Dual • Direct sum • Function space • Quotient • Subspace • Tensor product
Numerical: Floating-point • **Numerical stability** • Basic Linear Algebra Subprograms • Sparse matrix • Comparison of linear algebra libraries
Category

Category: Numerical analysis