1. Well-conditioned Problem

In calculus, we have learned the notion of continuity, that is, we say a function $f(x)$ is continuous at some point $x = a$ if and only if the following statement is true: for every $\epsilon > 0$, we can find a $\delta > 0$ such that

$$|f(x) - f(a)| < \epsilon$$

if $|x - a| < \delta$.

However, this is awfully abstract. What it really means is that given an input very close to $x = a$ (closeness measured by δ), then the output won't be also so far away from the true output (measured by ϵ).

This definition helps us establish something call **well-conditioned** problems. Suppose we are asked to solve a system of equations in matrix form, i.e.,

$$Ax = b.$$

We say this problem is **well-conditioned** if for every perturbation δA and δb, the solution $\tilde{x}$ to the perturbed problem

$$(A + \delta A)\tilde{x} = (b + \delta b)$$

is not too far off from the true solution x. In other words, this problem doesn't go crazy when nudged. Otherwise, we call the problem **ill-conditioned**.

More generally, one may think of a problem as a mapping from the data space to the solution space, more precisely,

$$f : \mathcal{D} \to \mathcal{S}.$$

In the example of a system of equations, the function is in fact, "2-dimensional",

$$f(A, b) = A^{-1}b = x$$

where we certainly seek the inverse of the matrix. With input data b and the known operations A, we seek a solution $x = A^{-1}b$. If f is "continuous" in both arguments, then we say this system of equation is a well-conditioned problem. This further suggests that the well-conditionedness of f depends on the invertibility of A.

2. Stability of Algorithms

Meanwhile, to solve the problem, one may use an algorithm, which is an approximation of the real problem. Though an approximation, an algorithm consumes inputs and produces outputs. We call an algorithm **stable** or **unstable**, if we insert perturbed inputs and see if we obtain minor or major fluctuations in outputs. We sometimes refer to inputs as **initial data**. Of course, not all algorithms are stable given any initial data. If one can find a condition on the initial data such that the algorithm is stable, then we say the algorithm is **conditionally stable**.

A more general way of looking at algorithms is to consider the same two spaces mentioned in the first section, but now, an approximation function

$$\hat{f} : \mathcal{D} \to \mathcal{S}.$$

One way of quantifying the stability of an algorithm is by looking at how errors are changing at different stages of the operations.

Definition. Suppose that $E_0 > 0$ denotes an error introduced at some stage in the calculations, and E_n represents the magnitude of the error after n subsequent operations.

(1) (linear) If $E_n \approx CnE_0$ where C is independent of n, then we say the growth of error is **linear**.
(2) (exponential) If $E_n \approx C^n E_0$ for some $C > 1$, then the growth of error is **exponential**.

Example. (Unstable Sequence) Consider the sequence of numbers (possibly representing an iterative algorithm)

$$p_n = c_1 \left(\frac{1}{3}\right)^n + c_2 3^n$$

which solves the recurrence relation

$$p_n = \frac{10}{3} p_{n-1} - p_{n-2}, \quad n = 2, 3, \ldots.$$

with two degrees of freedom, c_1 and c_2 (think of them as the discrete version of "integration constants" of a second-order ODE). They are determined if we specify p_0 and p_1, that is, the "initial conditions". The way to solve second order recurrence relation like the above is to guess first a solution of the form

$$p_n = c_1 r_1^n + c_2 r_2^n.$$

Plugging these in, we have

$$c_1 r_1^n + c_2 r_2^n = \frac{10}{3}\left(c_1 r_1^{n-1} + c_2 r_2^{n-1}\right) - \left(c_1 r_1^{n-2} + c_2 r_2^{n-2}\right).$$

Collecting similar terms, we have

$$c_1 r_1^{n-2}\left(r_1^2 + \frac{10}{3} r_1 - 1\right) + c_2 r_2^{n-2}\left(r_2^2 + \frac{10}{3} r_2 - 1\right) = 0.$$

This implies that r_1 and r_2 must be the solution to the quadratic equation

$$r^2 + \frac{10}{3} r - 1 = 0.$$

Suppose $p_0 = 1$ and $p_1 = \frac{1}{3}$. Then, a straightforward calculation yields

$$c_1 = 1, \quad c_2 = 0$$

which implies, with the specific initial conditions, $p_n = \left(\frac{1}{3}\right)^n$ is the unique solution.

Now, suppose we have used five-digit rounding to compute the sequence given by $p_n = \left(\frac{1}{3}\right)^n$, that is, we begin with $\widehat{p_0} = 1.0000$ and $\widehat{p_1} = 0.33333$ – this first readily modifies the constants $\widehat{c_1} = 1.0000$ and $\widehat{c_2} = -0.12500 \times 10^{-5}$. Let's check this.

$$1.0000 = \widehat{p_0} = \widehat{c_1} + \widehat{c_2}$$

$$0.33333 = \widehat{p_1} = \frac{1}{3}\widehat{c_1} + 3\widehat{c_2}$$

which yields a system of equations for $\widehat{c_1}$ and $\widehat{c_2}$. Using a substitution derived from the first equation, we have

$$\widehat{c_1} = 1.0000 - \widehat{c_2} \implies 0.33333 = (1.0000 - \widehat{c_2})\frac{1}{3} + \widehat{c_2}\,(3)$$

and thus

$$\left(3 - \frac{1}{3}\right)\widehat{c_2} = 0.33333 - \frac{1}{3} \implies \widehat{c_2} = \frac{0.33333 - \frac{1}{3}}{3 - \frac{1}{3}} = -0.12499\ldots \times 10^{-5}.$$

The five-digit rounding for $\widehat{c_2}$ is -0.12500×10^{-5}, while $\widehat{c_1} = 1.0000 - \widehat{c_2} = 1.0000 - (-0.00000125) = 1.00000125$ which rounds to 1.0000 (note the leading 1 ate up the decimals).

Thus, the sequence $\widehat{p_n}$ generated by these rounded numbers is

$$\widehat{p_n} = 1.0000 \left(\frac{1}{3}\right)^n - 0.12500 \times 10^{-5} \left(3\right)^n,$$

which has round-off error

$$p_n - \widehat{p_n} = \underbrace{0.12500 \times 10^{-5}}_{E_0} \underbrace{(3)^n}_{C^n}.$$

According to our definition of a stable/unstable algorithm, we observe that this procedure is **unstable** because the error grows exponentially with n.

This also informs us that solutions on paper to a problem may not be accurately evaluated by a machine. Finding an analytical formula is nice. But to put it to practice, one must be careful with the algorithm, sometimes, as simple as "plugging numbers in".

Remark. Algorithms that give linear errors are stable. Algorithms that give exponential errors are unstable.