Condition and stability

See Section 1.3 of the textbook for further discussion.

Conditioning

Suppose we wish to solve a problem with an input x and output $f(x)$.

Conditioning: A problem is called **well-conditioned** if small changes in the input lead to small changes in the output. Otherwise, the problem is called **ill-conditioned**.

That is, a well-conditioned problem is one where the output is not too sensitive to the input. An ill-conditioned problem is sensitive; a small error made in the input can lead to a drastic difference in the output.

Generally speaking, it is challenging to solve ill-conditioned problems numerically, because any errors we introduce will get amplified. Because the poor condition is inherent to the problem, there is sometimes no easy way around it.

For example, consider the problem of evaluating

$$f(x) = \tan x$$

for x near $\pi/2$. Suppose, say, we take

$$x_1 = \pi/2 - 0.001, \quad x_2 = \pi/2 - 0.002.$$

Then

$$|x_1 - x_2| = 0.001, \quad |f(x_1) - f(x_2)| = 500$$

so the small difference in the x-values leads to large differences in f.

Why does this happen? Since

$$f(x + \Delta x) \approx f(x) + f'(x)\Delta x \implies f(x + \Delta x) - f(x) \approx f'(x)\Delta x$$

we see that if f' is large, the problem of evaluating $f(x)$ can be ill-conditioned; errors get amplified by a factor of $f'(x)$. For the example, $f'(x) \to \infty$ as $x \to \pi/2$.

Note that if we instead want to consider the growth of *relative errors* then

$$\frac{f(x + \Delta x) - f(x)}{f(x)} = \left(\frac{xf(x)}{f'(x)}\right)\frac{\Delta x}{x}$$

so the relative error is amplified by a factor of xf/f'. We call this value the **condition number** for the problem, which measures the growth of relative errors from input to output.

Stability

An algorithm is said to be **(numerically) stable** if small errors in the inputs and at each step lead to small errors in the solution. Note that this is a property of the algorithm, while the 'condition' is a property of the problem itself.

Typically, we want to find stable numerical algorithms to solve well-conditioned problems. Because of the presence of errors, too much amplification of errors by the algorithm will render it useless! Some amount of growth, however, is unavoidable.

Suppose the initial error is e_0 and the error after n steps is e_n. If $e_n = O(ne_0)$, that is acceptable (after all, accumulation of rounding errors will cause this growth anyway, and that cannot be avoided). On the other hand, if e_n grows exponentially with n then that can be a disaster.

An algorithm that amplifies errors (too much) is called **unstable**. Let's look at a somewhat contrived example. Suppose we wish to solve a problem where the solution is

$$a_n = 2^{-n}.$$

and the method used to compute it is the recurrence

$$a_{n+1} = \frac{5}{2}a_n - a_{n-1}, \qquad a_0 = 1, \quad a_1 = 1/2.$$

It is easy to check that the solution is $a_n = 2^{-n}$. If all calculations are done exactly, then we will get the sequence. In this case, all arithmetic is done exactly (division/multiplication by 2 is exact) and the computed solution is 2^{-n} (exactly).

Suppose now that there is an initial error:

$$\tilde{a}_0 = 1 + \delta, \quad \tilde{a}_1 = 1/2$$

where $\delta \approx 2 \cdot 10^{-16}$ (on the order of rounding error). Running the following code in Matlab,

```
A(1) = 1 + delta; A(2) = 1/2;
for i=3:N
    A(i+1) = 5/2*A(i) - A(i-1);
end
```

leads to disaster (for any δ with $1 + \delta \neq 1$). Eventually, the computed solution begins to grow exponentially - even if the initial error is on the order of 10^{-16} and we have only done about 200 operations. The issue is that there is a hidden instability in the recurrence! The exact solution to the recurrence for $\tilde{a}_n$ is actually

$$a_n = \left(1 + \frac{4\delta}{3}\right) 2^{-n} - \frac{\delta}{3}2^n.$$

There is an exponential term that has nothing to do with the solution we want, and

$$a_n \sim -\frac{\delta}{3}2^n \text{ as } n \to \infty.$$

If $2^{-n} \sim \delta$ (which occurs only after about 50 iterations) then the spurious 2^n term will overpower the part that we want.