# Numerics

## 1. Errors

We have run into numerical errors already in our discussion of the expression of numbers on computers. Now let us discuss this issue more completely.

There different sorts of errors in code. They are worth reviewing in order to clarify what sorts of things we typically worry about:

- Bugs: Just mistakes in your code, typos, etc. It is worth mentioning at this stage the sorts of strategies that can be taken to minimize bugs in software. The simplest bugs are the ones that cause your software to fail completely; you just need to fix the problem. More complicated are bugs which cause your answer to be wrong. These require explicitly written tests to find systematically. Two general classes of tests exist:
  - Unit tests: Typically the test of a single function or component of the code. Tests that its output and behavior conforms to expectations given a controlled input.
  - Functional tests: Tests that a the software as a whole is doing the correct thing. In the context of computational physics, a useful functional test will involve a calculation for which an analytic solution exists (e.g. a spherically symmetric calculation with a full 3D code could be a functional test) or for which scaling relations or conservation laws can be tested.

  I will show in the notebook some very simplified examples of these sorts of test.
- Random errors: Sometimes memory or disk can be corrupted, and truly random errors can occur. These are rare. If there is random variation of your output with precisely the same input, that is happening commonly, then likely you have a *bug*.
- Approximation errors: Sometimes your calculation is approximate more-or-less on purpose. For each problem, you must be able to calculate some estimate of how wrong it is *supposed* to be given the approximations.
- Round-off errors: These are the errors we have discussed already, due to the finite precision of numbers on computers.

## 2. Subtractive cancellation errors

We saw before that additions of many small numbers can lead to numerical instability. Another, even more common, instance of instability is when two large numbers are subtracted to yield a relatively smaller number. This is problematic because the finite precision of the large numbers means their remainder is, relatively speaking, lower precision.

For example, take the calculation $a = b - c$. Each number carries some fractional error $\epsilon$:

$$a_c = b_c - c_c = (b + \epsilon_b) - (c + \epsilon_c) \tag{1}$$

This means that:

$$\epsilon_a = a_c - a = \epsilon_b - \epsilon_c \tag{2}$$

These two terms don't necessarily cancel, and half the time will compound each other. So if $b/a$ and $c/a$ are large, $\epsilon_a$ can be very large.

## 3. Error propagation

Let me write the error in $a$ as $\epsilon_a = a_c - a$, and similarly for other variables. I show in the notebook what the distribution of $\epsilon_a$ looks like by approximating $a$ with a 64-bit float and looking at the distribution of $a_c$ if it is expressed as a 32-bit float.

Often when we talk about the "error" in a quantity, we really mean the "uncertainty." And what we usually mean by that is the distribution of the errors. We would write $\delta a$ is a measure — the standard deviation, specifically — of the distribution of $\epsilon_a$. The variance of $\epsilon_a$ around its mean (zero) is:

$$\delta_a^2 = \langle \epsilon_a^2 \rangle \approx \frac{1}{N} \sum_i \epsilon_{a,i}^2 \tag{3}$$

where $\langle\rangle$ indicates an average over many instances, which we can estimate with the last expression above. The standard deviation is the square root of the variance.

**Walk me through the calculation of the standard deviation of a uniform distribution of $\epsilon$ between 0 and 1.**

**What is the standard deviation of a Gaussian?**

When I add, say, $a_c$ and $b_c$, then I can evaluate variance in the result as follows:

$$\begin{aligned} \left[\delta_{(a+b)}\right]^2 &= \left\langle [\epsilon_a + \epsilon_b]^2 \right\rangle \\ &\approx \left\langle \left[\epsilon_a^2 + \epsilon_b^2 + 2\epsilon_a\epsilon_b\right] \right\rangle \\ &= \left\langle \left[\epsilon_a^2 + \epsilon_b^2\right] \right\rangle \\ &= \delta_a^2 + \delta_b^2 \end{aligned} \tag{4}$$

Where, very importantly, we use the fact that $\epsilon_a$ and $\epsilon_b$ are (probably) statistically uncorrelated.

When I perform some operation $f()$ on $a_c$, then I can evaluate variance in the result as follows:

$$\left\langle [\delta f(a)]^2 \right\rangle \approx \left\langle \left[\frac{\partial f}{\partial a}\right]^2 \delta a^2 \right\rangle \tag{5}$$

and the standard deviation should be the square root of this.

Then when the function involves two uncorrelated variables:

$$\left\langle [\delta f(a,b)]^2 \right\rangle \approx \left\langle \left[\frac{\partial f}{\partial a}\right]^2 \delta a^2 + \left[\frac{\partial f}{\partial b}\right]^2 \delta b^2 \right\rangle \tag{6}$$

because (as before) the terms involving $\delta a \delta b$ vanish in the expectation value.

Note how the above expression works for $f(a,b) = a + b \to \delta f^2 = \delta a^2 + \delta b^2$. If we have a sum with $N$ terms $a_i$, all with similar $\delta a_i$, then

$$\delta f^2 = \sum_{i=1}^{N} \delta a_i^2 \sim N \delta a^2 \tag{7}$$

or in other words the round-off for a series of $N$ operations scales as $\sqrt{N}$.

## 4. Approximation Error

The book describes a particular example of approximation error. I'll describe a different example.

If we want to approximate the integral of a function, a simple thing to do is to just sum a bunch of small intervals:

$$\int_{x_0}^{x_1} \mathrm{d}x\, f(x) \approx \sum_i \Delta x f(x_i) \tag{8}$$

This is just the same as taking the definition of the integral, but not taking the limit $\Delta x \to 0$.

We can take some simple function, e.g.:

$$f(x) = x^2 - x^3 + x^4 \tag{9}$$

with a known integral:

$$F(x) = \frac{x^3}{3} - \frac{x^4}{4} + \frac{x^5}{5} \tag{10}$$

and we can approximate the integral using this piecewise constant method.

In this case the error for $N$ steps is:

$$\epsilon = \left( \sum_{i=0}^{N-1} \Delta x f(x_i) \right) - F(x) \tag{11}$$

This error is the approximation error.

As you can see in the notebook, this goes down as you increase the number of steps, because it is becoming a better and better approximation.

However, it requires a greater and greater number of computations. So for a large enough number of computations, the round-off error can start to creep up. In what we've done here, this matters at 32-bit precision but not at 64-bit precision (where for the range we pick the round-off is not yet important). So above a certain $N$, a few hundred thousand in this case, the decrease in approximation error no longer offsets the increase in round-off error; at best your final number is at machine precision, but in fact it starts to degrade from that!

You can even calculate this from taking the approximate machine precision of each summation $\sqrt{N}\delta$q.