# Week 1: Monday, Jan 23

## Safe computing

If this class were about shooting rather than computing, we'd probably start by talking about where the safety is. Otherwise, someone would shoot themself in the foot. The same is true of basic error analysis: even if it's not the most glamorous part of scientific computing, it comes at the start of the course in order to keep students safe from themselves. For this, a rough-and-ready discussion of sensitivity analysis, simplified models of rounding, and exceptional behavior is probably sufficient. We'll go a bit beyond that, but you can think of most of the material in the first week in this light: basic safety training for computing with real numbers.

## Some basics

Suppose $\hat{x}$ is an approximation for $x$. The *absolute error* in the approximation is $\hat{x} - x$; the *relative error* is $(\hat{x} - x)/x$. Both are important, but we will use relative error more frequently for this class.

Most measurements taken from the real world have error. Computations with approximate inputs generally produce approximate outputs, even when the computations are done exactly. Thus, we need to understand the *propagation* of errors through a computation. For example, suppose that we have two inputs $\hat{x}$ and $\hat{y}$ that approximate true values $x$ and $y$. If $e_x$ and $\delta_x = e_x/x$ are the absolute and relative errors for $x$ (and similarly for $y$), we can write

$$\begin{aligned}\hat{x} &= x + e_x = x(1 + \delta_x) \\ \hat{y} &= y + e_y = y(1 + \delta_y).\end{aligned}$$

When we add or subtract $\hat{x}$ and $\hat{y}$, we add and subtract the absolute errors:

$$\begin{aligned}\hat{x} + \hat{y} &= (x + y) + (e_x + e_y) \\ \hat{x} - \hat{y} &= (x - y) + (e_x - e_y).\end{aligned}$$

Note, however, that the relative error in the approximation of $x - y$ is $(e_x - e_y)/(x - y)$; and if $x$ and $y$ are close, this relative error may be much larger than $\delta_x$ or $\delta_y$. This effect is called *cancellation*, and it is a frequent culprit in computations gone awry.

From the perspective of relative error, multiplication and division are more benign than addition and subtraction. The relative error of a product is (approximately) the sum of the relative errors of the inputs:

$$\begin{aligned}\hat{x}\hat{y} &= xy(1+\delta_x)(1+\delta_y) \\ &= xy(1+\delta_x+\delta_y+\delta_x\delta_y) \\ &\approx xy(1+\delta_x+\delta_y).\end{aligned}$$

If $\delta_x$ and $\delta_y$ are small, then $\delta_x\delta_y$ is absolutely miniscule, and so we typically drop it when performing an error analysis[1] Similiarly, using the fact that

$$(1+\delta)^{-1} = 1 - \delta + O(\delta^2),$$

the relative error in a quotient is approximately the difference of the relative errors of the inputs:

$$\begin{aligned}\frac{\hat{x}}{\hat{y}} &= \frac{x}{y}\left(\frac{1+\delta_x}{1+\delta_y}\right) \\ &\approx \frac{x}{y}(1+\delta_x)(1-\delta_y) \\ &\approx \frac{x}{y}(1+\delta_x-\delta_y).\end{aligned}$$

More generally, suppose $f$ is some differentiable function of $x$, and we approximate $z = f(x)$ by $\hat{z} = f(\hat{x})$. We estimate the absolute error in the approximation by Taylor expansion:

$$\hat{z} - z = f(\hat{x}) - f(x) \approx f'(x)(\hat{x} - x).$$

The magnitude $|f'(x)|$ tells us the relationship between the size of absolute errors in the input and the output. Frequently, though, we care about how *relative* errors are magnified by the computation:

$$\frac{\hat{z}-z}{z} \approx \left(\frac{xf'(x)}{z}\right)\frac{\hat{x}-x}{x}.$$

The size of the quantity $xf'(x)/f(x)$ that tells us how relative errors on input affect the relative errors of the output is sometimes called the *condition number* for the problem. Problems with large condition numbers ("ill-conditioned" problems) are delicate: a little error in the input can destroy the accuracy of the output.

[1] This is a *first-order* error analysis or *linearized* error analysis, since we drop any terms that involve products of small errors.

## A simple example

What's the average velocity of the earth as it travels around the sun?

Yes, you can ask Google this question, and it will helpfully tell you that the answer is about 29.783 km/s. But if you have a few facts at your fingertips, you can get a good estimate on your own:

- The earth is $t \approx 500$ light-seconds from the sun.
- The speed of light is $c \approx 3 \times 10^8$ m/s.
- A year is about $y \approx \pi \times 10^7$ seconds.

So the earth follows a roughly circular orbit with radius of about $r = ct \approx 1.5 \times 10^{11}$ meters, traveling $2\pi r$ meters in a year. So our velocity is about

$$\frac{2\pi ct}{y} \approx \frac{2\pi \times 1.5 \times 10^{11} \text{ m}}{\pi \times 10^7 \text{ s}} = 30 \text{ km/s}.$$

How wrong is the estimate $\hat{v} = 30$ km/s? The absolute error compared to the truth ($v = 29.783$ km/s a la Wiki answers) is about 217 m/s. Unfortunately, it's hard to tell without context whether this is a good or a bad error. Compared to the speed of the earth going around the sun, it's pretty small; compared to my walking speed, it's pretty big. It's more satisfying to know the relative error is

$$\frac{\hat{v} - v}{v} \approx 0.0073,$$

or less than one percent. Informally, we might say that the answer has about two correct digits.

What is the source of this slightly-less-than-a-percent error? Crudely speaking, there are two sources:

1. We assumed the earth's orbit is circular, but it's actually slightly elliptical. This *modeling error* entered before we did any actual calculations.
2. We approximated all the parameters in our calculation: the speed of light, the distance to the sun in light-seconds, and the length of a year. More accurate values are:

$$\begin{aligned} t &= 499.0 \text{ s} \\ c &= 299792458 \text{ m/s} \\ y &= 31556925.2 \text{ s} \end{aligned}$$

> If we plug these values into our earlier formula, we get a velocity of 29.786 km/s. This is approximated by 30 km/s with a relative error of 0.0072.

We'll put aside for the moment the ellipticity of the earth's orbit (and the veracity of Wiki Answers) and focus on the error inherited from the parameters. Denoting our earlier estimates by $\hat{t}$, $\hat{c}$, and $\hat{y}$, we have the relative errors

$$\begin{aligned}
\delta_t &\equiv (\hat{t} - t)/t = 20 \times 10^{-4} \\
\delta_c &\equiv (\hat{c} - c)/c = 7 \times 10^{-4} \\
\delta_y &\equiv (\hat{y} - t)/y = -45 \times 10^{-4}
\end{aligned}$$

According to our error propagation formulas, we should have

$$\frac{2\pi\hat{c}\hat{t}}{\hat{y}} \approx \left(\frac{2\pi ct}{y}\right)(1 + \delta_c + \delta_t - \delta_y).$$

In fact, this looks about right:

$$20 \times 10^{-4} + 7 \times 10^{-4} + 45 \times 10^{-4} = 72 \times 10^{-4}.$$

## An example of error amplification

Our previous example was benign; let us now consider a more sinister example. Consider the integrals

$$E_n = \int_0^1 x^n e^{x-1}\,dx$$

It's easy to find $E_0 = 1 - 1/e$; and for $n > 0$, integration by parts[2] gives us

$$E_n = x^n e^{x-1}\big|_0^1 - n\int_0^1 x^{n-1}e^{x-1}\,dx = 1 - nE_{n-1}.$$

We can use the following MATLAB program to compute $E_n$ for $n = 1, \ldots, n_{\max}$:

[2] If you do not remember integration by parts, you should go to your calculus text and refresh your memory.

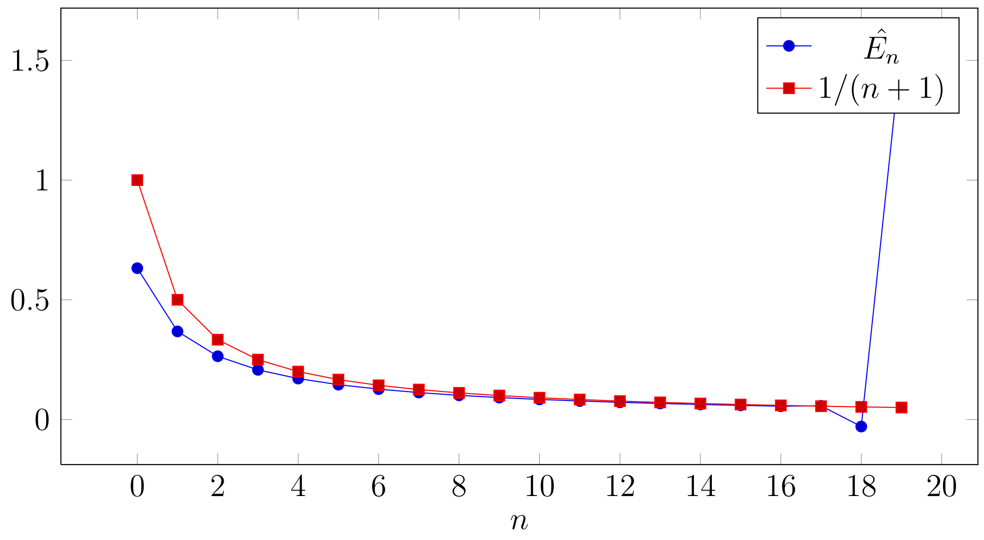

Figure 1: Computed values $\hat{E}_n$ compared to a supposed upper bound.

```
%
% Compute the integral of x^n exp(x-1) from 0 to 1
% for n = 1, 2, ..., nmax
%
function E = lec01int(nmax)

  En = 1-1/e;
  for n = 1:nmax
    En = 1-n*En;
    E(n) = En;
  end
```

The computed values of $\hat{E}_n$ according to our MATLAB program are shown in Figure 1. A moment's contemplation tells us that something is amiss with these problems. Note that $1/e \leq e^{x-1} \leq 1$ for $0 \leq x \leq 1$, so that

$$\frac{1}{e(n+1)} \leq E_n \leq \frac{1}{n+1}$$

As shown in the figure, our computed values violate these bounds. What has gone wrong?

The integral is perfectly behaved; the problem lies in the algorithm. The initial value $E_0$ can only be represented approximately on the computer; let us write the stored value as $\hat{E}_0$. Even if this were the only error, we would find the error in later calculations grows terribly rapidly:

$$\begin{aligned}
\hat{E}_0 &= E_0 + \epsilon \\
\hat{E}_1 &= 1 - \hat{E}_0 = E_1 - \epsilon \\
\hat{E}_2 &= 1 - 2\hat{E}_1 = E_2 + 2\epsilon \\
\hat{E}_3 &= 1 - 3\hat{E}_2 = E_3 - 3 \cdot 2\epsilon \\
&\vdots \\
\hat{E}_n &= E_n + (-1)^n n!\epsilon.
\end{aligned}$$

This is an example of an *unstable* algorithm.

## Sources of error

We understand the world through simplified models. These models are, from the outset, approximations to reality. To quote the statistician George Box:

> All models are wrong. Some models are useful.

While modeling errors are important, we can rarely judge their quality without the context of a specific application. In this class, we will generally take the model as a given, and analyze errors from the input and the computation.

In addition to error intrinsic in our models, we frequently have error in input parameters derived from measurements. In *ill-conditioned* problems, small relative errors in the inputs can severely compromise the accuracy of outputs. For example, if the input to a problem is known to within a millionth of a percent and the condition number for the problem is $10^7$, the result might only have one correct digit. This is a difficulty with the problem, not with the method used to solve the problem. Ideally, a well-designed numerical routine might warn us when we ask it to solve an ill-conditioned problem, but asking for an accurate result may be asking too much. Instead, we will seek *stable* algorithms that don't amplify the error undeservedly. That is, if the condition number for a problem is $\kappa$, a stable algorithm should amplify the input error by at most $C\kappa$, where $C$ is some modest constant that does not depend on the details of the input.

In order to produce stable algorithms, we must control the sources of error from within the computation itself. In the order we encounter them, these include:

**Roundoff error:** IEEE floating point arithmetic is essentially binary scientific notation. The number 1/3 cannot be represented exactly as a finite decimal. It can't be represented exactly in a finite number of binary digits, either. We can, however, approximate 1/3 to a very high degree of accuracy.

**Termination of iterations:** Nonlinear equations, optimization problems, and even linear systems are frequently solved by iterations that produce successively better approximations to a true answer. At some point, we decide that we have an answer that is good enough, and stop.

**Truncation error:** We frequently approximate derivatives by finite differences and integrals by sums. The error in these approximations is frequently called truncation error.

**Stochastic error:** Monte Carlo methods use randomness to compute approximations. The variance due to randomness typically dominates the error in these methods.