

TMA4125 Matematikk 4N

#3b – Numerical Errors

Ronny Bergmann

Institute of Mathematical Sciences, NTNU.

January 27, 2023



Introductory example: Computing $e \approx 2.71828182845$

We know from one of the last lectures that

$$\lim_{h \rightarrow 0} (1 + h)^{\frac{1}{h}} = e$$

So let's compare `np.exp(1)` to this limiting process – numerically

Numbers on the computer: Single precision

The usual representation of numbers: Float32 using 32 bit:

- ▶ 1 bit for the **sign**
- ▶ 23 bit for a fraction number, e. g. 0.5625, called **mantissa**
- ▶ 8 bit to an **exponent** (representing a factor $2^{-126}, \dots, 2^{127}$)

23 bit represent a range from 6 to 9 decimal digits.

⇒ Fixing the exponent, we have different precision limitations

- ▶ on $[1, 2]$: the next number after 1 is $1 + 2^{-23} \approx 1 + 1.192093 \cdot 10^{-7}$
- ▶ on $[2^{23}, 2^{24}]$, i. e. between $8\,388\,608 = 2^{23}$ and $16\,777\,216 = 2^{24}$: the next larger number is a step of $2^0 = 1$.
- ▶ ...similarly for smaller numbers the “steps” are also smaller.
- ▶ on “smallest scale”: 0 is followed by $2^{-126} \approx 1.175494 \cdot 10^{-38}$

Underflow, overflow, rounding

If we obtain a result in a calculation, that is smaller than the smallest value we can represent, this is called **arithmetic underflow**. These are set to zero.

Example. In `Float32` this affects any number below $2^{-126} \approx 1.175 \cdot 10^{-38}$. What can we do if we encounter this problem? Rescale.

If we obtain a result in a calculation that is larger than the largest value we can represent, this is called **arithmetic overflow**,
⇒ This results in an error, `NaN` (Not-a-Number), sometimes `Inf` (depending on the language).

Otherwise we **round** the result of a computation to the nearest number we can represent, leading to **rounding errors**.

Rounding errors and machine precision

An important measure for exactness is then the **relative rounding error**. If we round x to \bar{x} then the **relative rounding error** is defined as

$$\left| \frac{x - \bar{x}}{x} \right|$$

One can show that the relative error in floating point operations is

$$\left| \frac{x - \bar{x}}{x} \right| \leq \varepsilon = \frac{1}{2} b^{1-m},$$

where m is the length of the **mantissa** and on the computer $b = 2$. The number ε is called **machine precision**.

⇒ in **single precision** (32 bit): $m = 23$ and hence $\varepsilon \approx 5.96 \cdot 10^{-8}$

⇒ in **double precision** (64 bit): $m = 53$ and hence $\varepsilon \approx 1.11 \cdot 10^{-16}$

Often computations are performed in double precision.



NTNU

What are general Ideas to avoid rounding errors?

Linear systems of equations.

When solving a linear system

$$Ax = b$$

we speak of an **ill-conditioned** system if a small change in b (the “input”) results in a large change in the “output” x .

Example. This happens for the **Vandermonde-Matrix**

$$V = \begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^n \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{pmatrix}$$

when two nodes x_i, x_j are close together.

More generally: If two columns are nearly linear dependent.

Linear systems and error propagation

Error propagation refers to the (schematic) investigation of How does an error in the “input” affect the result “output”?

For our linear system $Ax = b$ we want know how “far away” is the solution for an error on the right hand side

$$A\tilde{x} = b + e, \quad \text{where we know an upper bound } \|e\| \leq \delta$$

More precisely: How much is such an error *emphasized*?

Formally:

What is the relation of the relative input error $\frac{\|e\|}{\|b\|}$ to the relative output error $\frac{\|x - \tilde{x}\|}{\|x\|}$?

A very simple example is

$$A = \begin{pmatrix} \frac{1}{4} & 0 \\ 0 & 4 \end{pmatrix}, \mathbf{b} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \text{ and } \|e\| \leq 0.1$$

The error might get emphasized by a factor of 16 !

The condition number.

We saw in the example the “worst case scenario” (largest factor). We can formalize this if A is invertible as the **condition number**

$$\kappa(A) = \max_{\mathbf{e}, \mathbf{b} \neq 0} \frac{\frac{\|A^{-1}\mathbf{e}\|}{\|A^{-1}\mathbf{b}\|}}{\frac{\|\mathbf{e}\|}{\|\mathbf{b}\|}} = \max_{\mathbf{e} \neq 0} \frac{\|A^{-1}\mathbf{e}\|}{\|\mathbf{e}\|} \max_{\mathbf{b} \neq 0} \frac{\|\mathbf{b}\|}{\|A^{-1}\mathbf{b}\|}$$

Since $\mathbf{x} = A^{-1}\mathbf{b}$ we can write this as

$$\max_{\mathbf{e} \neq 0} \|A^{-1}\mathbf{e}\| \|\mathbf{e}\| \max_{\mathbf{x} \neq 0} \frac{A\|\mathbf{x}\|}{\mathbf{x}} = \|A^{-1}\| \cdot \|A\|,$$

where these last two norm for the matrix used is the **spectral norm**, defined for some matrix C as

$$\|C\| = \sqrt{\lambda_{\max}(C^T C)},$$

i.e. the largest Eigenvalue of $C^T C$ or the spectral radius of C .

A(n artificial) polynomial

If we consider

$$p_1(x) = -128 + 1344x - 6048x^2 + 15120x^3 - 22680x^4 + 20412x^5 - 10206x^6 + 2187x^7$$

We could use Python to plot the function to find the roots $p_1(x^*) \stackrel{!}{=} 0$

This is not numerically stable on – say $[\frac{6}{10}, \frac{7}{10}]$. Any idea why?

We can also write this a little simpler as

$$p_1(x) = (3x - 2)^7$$

where we also already see, that the only zero is at $x^* = \frac{2}{3}$. Let's look at the plot of this.

this second form also avoids adding, but especially subtracting numbers of approximately equal size (subtraction cancellation)!

Forward and central differences

For some fixed h we denote the forward difference by

$$\Delta u(x) = u(x+h) - u(x)$$

And we already saw that the **error by rounding** is the size of **machine precision** ε “on every scale”.

We obtain on the computer $\tilde{\Delta}u(x)$ given by

$$\tilde{\Delta}u(x) = \Delta u(x) \pm \varepsilon|u(x)| = u(x+h) - u(x) \pm \varepsilon|u(x)|.$$

Plugging this into the **forward finite difference** (see first week) we get

$$\frac{\tilde{\Delta}u(x)}{h} = \frac{u(x+h) - u(x) \pm \varepsilon|u(x)|}{h} = \frac{u(x+h) - u(x)}{h} \pm \frac{\varepsilon|u(x)|}{h}.$$

For small h the last term will **emphasize** the error term.

Rounding Error Analysis on Finite Differences

Question. When will the rounding error be of **same magnitude** as the approximation error?

In other words: When the rounding error “dominates” the error happening, there is no use in decreasing h further.

Combining the rounding error with the approximation error, we obtain

$$h \approx \sqrt{\left| \frac{u(x)}{C} \right| \varepsilon} \quad \text{or in short} \quad h = \mathcal{O}(\sqrt{\varepsilon})$$

which in our case is $\sqrt{\varepsilon} = \sqrt{10^{-16}} = 10^{-8}$.

Analogously. For the central difference: $h = \mathcal{O}(\sqrt[3]{\varepsilon})$, i. e. $\sqrt[3]{\varepsilon} \approx 10^{-5.33}$.

Summary

When we have numerical (rounding errors) they might come from

- ▶ implementation detail \Rightarrow Rephrase/Rewrite your code called **numerical instability**
 \Rightarrow avoid them by rephrasing/reimplementing your algorithm
- ▶ inherently from the problem,
e. g. the condition number of the matrix.
usually called **ill-conditioned** or **mathematical instability**.

Due to the limits of **machine precision** when representing numbers.