Contents I

Positional Notation

Definition 1 (Positional Notation)

System representing numbers $x \in \mathbb{R}$ using:

$$\begin{aligned} x &= \pm \dots m_2\beta^2 + m_1\beta + m_0 + m_{-1}\beta^{-1} + m_{-2}\beta^{-2} \dots \\ &= \sum_{i \in \mathbb{Z}} m_i \beta^i \end{aligned}$$

$\beta \in \mathbb{N}, \beta \geq 2$, is called *base*,
$m_i \in \{0, 1, 2, \dots, \beta - 1\}$ are called *digits*

History:

- Babylonians (≈ -1750), $\beta = 60$
- Base 10 from ~ 1580
- Pascal: all values $\beta \geq 2$ may be used

Fixed-Point Numbers

Fixed-point numbers: truncate series after finite number of terms

$$x = \pm \sum_{i=-k}^{n} m_i \beta^i$$

Problem: scientific applications use numbers of very different orders of magnitude

$$\begin{aligned} \text{Planck constant:} &\quad 6.626093 \cdot 10^{-34}\ \text{Js} \\ \text{Avogadro constant:} &\quad 6.021415 \cdot 10^{23}\ \tfrac{1}{\text{mol}} \\ \text{Electron mass:} &\quad 9.109384 \cdot 10^{-31}\ \text{kg} \\ \text{Speed of light:} &\quad 2.997925 \cdot 10^{8}\ \tfrac{\text{m}}{\text{s}} \end{aligned}$$

Floating-point numbers can represent all such numbers with acceptable accuracy

Floating-Point Numbers

Definition 2 (Floating-Point Numbers)

Let $\beta, r, s \in \mathbb{N}$ and $\beta \geq 2$. The *set of floating-point numbers* $\mathbb{F}(\beta, r, s) \subset \mathbb{R}$ consists of all numbers with the following properties:

1. $\forall x \in \mathbb{F}(\beta, r, s)\colon x = m(x) \cdot \beta^{e(x)}$ with

$$m(x) = \pm \sum_{i=1}^{r} m_i \beta^{-i}, \quad e(x) = \pm \sum_{j=0}^{s-1} e_j \beta^j$$

   with digits m_i and e_j.
   m is called *mantissa*, e is called *exponent*.

2. $\forall x \in \mathbb{F}(\beta, r, s)\colon x = 0 \vee m_1 \neq 0$. This is called *normalization* and makes the representation unique.

Example

Example 3

1. $\mathbb{F}(10,3,1)$ consists of the numbers

$$x = \pm(m_1 \cdot 0.1 + m_2 \cdot 0.01 + m_3 \cdot 0.001) \cdot 10^{\pm e_0}$$

with $m_1 \neq 0 \vee (m_1 = m_2 = m_3 = 0)$, e.g., 0, $0.999 \cdot 10^4$, and $0.123 \cdot 10^{-1}$, but *not* $0.140 \cdot 10^{-10}$ (exponent to small)

2. $\mathbb{F}(2,2,1)$ consists of the numbers

$$x = \pm(m_1 \cdot 0.5 + m_2 \cdot 0.25) \cdot 2^{\pm e_0}$$

$$\Longrightarrow \mathbb{F}(2,2,1) = \left\{-\frac{3}{2}, -1, -\frac{3}{4}, -\frac{1}{2}, -\frac{3}{8}, -\frac{1}{4}, 0, \frac{1}{4}, \frac{3}{8}, \frac{1}{2}, \frac{3}{4}, 1, \frac{3}{2}\right\}$$

Standard: IEEE 754 / IEC 559

Goal: portability of programs with floating-point arithmetics
Finalized 1985

$\beta = 2$, with four levels of accuracy and normalized representation:

	single	single-ext	double	double-ext
e_{max}	127	$\geq$ 1024	1023	$\geq$ 16384
e_{min}	-126	$\leq$ -1021	-1022	$\leq$ -16381
Bits expon.	8	$\leq$ 11	11	$\geq$ 15
Bits total	32	$\geq$ 43	64	$\geq$ 79

The standard defines four kinds of rounding:
to $-\infty$, to $+\infty$, to 0, and to nearest

Since 2008: additionally half precision and quadruple precision

Double Precision

Let's have a closer look at double precision:

- 64 bit in total
- 11 bit for exponent, stored without sign as $c \in [1, 2046]$
- Let $e := c - 1023 \implies e \in [-1022, 1023]$, no sign necessary
- The values $c \in \{0, 2047\}$ are special:
  - $c = 0 \wedge m = 0$ encodes zero
  - $c = 0 \wedge m \neq 0$ encodes denormalized representation
  - $c = 2047 \wedge m = 0$ encodes ∞ (overflow)
  - $c = 2047 \wedge m \neq 0$ encodes NaN = "not a number", e.g., when dividing by zero

Double Precision

- $64 - 11 = 53$ bit for mantissa, one for sign, 52 bit remaining for mantissa digits
- $\beta = 2$ implies $m_1 = 1$
- This digit is called *hidden bit* and is never stored
- Therefore $r = 53$ in the sense of our definition of floating-point numbers

Double precision corresponds to $\mathbb{F}(2, 53, 10)$ + additional special codes.

Rounding Function

To approximate $x \in \mathbb{R}$ in $\mathbb{F}(\beta, r, s)$, we need a map

$$\mathrm{rd}\colon D(\beta, r, s) \to \mathbb{F}(\beta, r, s), \tag{1}$$

where $D(\beta, r, s) \subset \mathbb{R}$ is the domain containing $\mathbb{F}(\beta, r, s)$:

$$D := [X_-, x_-] \cup \{0\} \cup [x_+, X_+]$$

with $X_{+/-}$ being the numbers in $\mathbb{F}(\beta, r, s)$ with largest absolute value, and $x_{+/-}$ those with the smallest (apart from zero).
Note: this implies that x lies within the representable domain!

A reasonable demand is:

$$\forall x \in D\colon\ |x - \mathrm{rd}(x)| = \min_{y \in \mathbb{F}} |x - y|$$

(known as *best approximation property*)

Rounding Function

With $l(x) := \max\{y \in \mathbb{F} | y \leq x\}$ and $r(x) := \min\{y \in \mathbb{F} | y \geq x\}$ we have:

$$\mathrm{rd}(x) = \begin{cases} x & l(x) = r(x), x \in \mathbb{F} \\ l(x) & |x - l(x)| < |x - r(x)| \\ r(x) & |x - l(x)| > |x - r(x)| \\ ? & |x - l(x)| = |x - r(x)| \end{cases}$$

The last case requires further considerations. There are several possible choices.

Natural Rounding

Definition 4 (Natural Rounding)

Let $x = \mathrm{sign}(x) \cdot \left(\sum_{i=1}^{\infty} m_i \beta^{-i}\right) \beta^e$ the *normalized* representation of $x \in D$. Define

$$\mathrm{rd}(x) := \begin{cases} l(x) = \mathrm{sign}(x) \cdot \left(\sum_{i=1}^{r} m_i \beta^{-i}\right) \beta^e & \text{if } 0 \leq m_{r+1} < \beta/2 \\ r(x) = l(x) + \beta^{e-r} \text{ (last digit)} & \text{if } \beta/2 \leq m_{r+1} < \beta \end{cases}$$

This is the usual rounding everyone knows from school. It has the undesirable property of introducing bias, since rounding up is slightly more likely.
This is irrelevant in everyday life, but becomes important for small β, e.g., $\beta = 2$, and/or many operations (as in scientific computing).

Even Rounding

Definition 5 (Even Rounding)

Let (with notation as before)

$$\mathrm{rd}(x) := \begin{cases} l(x) & \text{if } |x - l(x)| < |x - r(x)| \\ l(x) & \text{if } |x - l(x)| = |x - r(x)| \wedge m_r \text{ even} \\ r(x) & \text{else} \end{cases}$$

This ensures that m_r in $\mathrm{rd}(x)$ is always even after rounding.

- For $\mathrm{rd}(x) = l(x)$ this is by definition.
- Else $\mathrm{rd}(x) = r(x) = l(x) + \beta^{e-r}$, m_r in $l(x)$ is odd, and addition of β^{e-r} changes the last digit by one.

This choice of rounding avoids systematic drift when rounding up, and corresponds to “round to nearest” in the standard.

Absolute and Relative Error

Definition 6 (Absolute and Relative Error)

Let $x' \in \mathbb{R}$ an approximation of $x \in \mathbb{R}$. Then we call

$$\Delta x := x' - x \quad \text{absolute error}$$

and for $x \neq 0$

$$\epsilon_{x'} := \frac{\Delta x}{x} \quad \text{relative error}$$

Rearranging leads to:

$$x' = x + \Delta x = x \cdot \left(1 + \frac{\Delta x}{x}\right) = x \cdot (1 + \epsilon_{x'})$$

Motivation

Motivation:

Let $\Delta x = x' - x = 100\,\text{km}$.
For $x =$ Distance Earth—Sun $\approx 1.5 \cdot 10^8\,\text{km}$,

$$\epsilon_{x'} = \frac{10^2\,\text{km}}{1.5 \cdot 10^8\,\text{km}} \approx 6.6 \cdot 10^{-7}$$

is relatively small.
But for $x =$ Distance Heidelberg—Paris $\approx 460\,\text{km}$,

$$\epsilon_{x'} = \frac{10^2\,\text{km}}{4.6 \cdot 10^2\,\text{km}} \approx 0.22 \quad (22\%)$$

is relatively large.

Error Estimation

Lemma 7 (Rounding Error)

When rounding in $\mathbb{F}(\beta, r, 2)$ the absolute error fulfills

$$|x - \mathrm{rd}(x)| \leq \frac{1}{2}\beta^{e(x)-r} \tag{2}$$

and the relative error (for $x \neq 0$)

$$\frac{|x - \mathrm{rd}(x)|}{|x|} \leq \frac{1}{2}\beta^{1-r}.$$

This estimate is sharp (i.e., the case "$=$" exists).
The number $\mathrm{eps} := \frac{1}{2}\beta^{1-r}$ is called *machine precision*.
$\mathrm{eps} = 2^{-24} \approx 6 \cdot 10^{-8}$ for single precision, and
$\mathrm{eps} = 2^{-53} \approx 1 \cdot 10^{-16}$ for double precision.

Floating-Point Arithmetics

We need arithmetics on $\mathbb{F}$:

$$\circledast \colon \mathbb{F} \times \mathbb{F} \to \mathbb{F} \quad \text{with} \quad \circledast \in \{\oplus, \ominus, \odot, \oslash\}$$

corresponding to the well-known operations $* \in \{+, -, \cdot, /\}$ on $\mathbb{R}$.

Problem: typically $x, y \in \mathbb{F} \not\Longrightarrow x * y \in \mathbb{F}$

Therefore the result has to be rounded. We *define*

$$\forall x, y \in \mathbb{F} \colon x \circledast y := \mathrm{rd}(x * y) \tag{3}$$

This guarantees “exact rounding”. The implementation of such a mapping is nontrivial!

Guard Digit

Example 8 (Guard Digit)

Let $\mathbb{F} = \mathbb{F}(10, 3, 1), x = 0.215 \cdot 10^8, y = 0.125 \cdot 10^{-5}$. We consider the subtraction $x \ominus y = \text{rd}(x - y)$.

1. Subtraction followed by rounding requires an extreme number of mantissa digits $\mathcal{O}(\beta^s)$!
2. Rounding before subtraction seems to produce same result. Good idea?
3. But: consider, e.g., $x = 0.101 \cdot 10^1, y = 0.993 \cdot 10^0$ $\implies$ relative error $18\% \approx 35$ eps
4. One, two additional digits are enough to achieve exact rounding!
5. These digits are called *guard digits* and are also used in practice (CPU), e.g., performing internal computations in 80 bit precision.

Table Maker Dilemma

Algebraic functions:
e.g., polynomials, $1/x$, $\sqrt{x}$, rational functions, ...
more or less: finite combination of basic arithmetic operations and roots

Transcendent functions:
everything else, e.g., $\exp(x), \ln(x), \sin(x), x^y, \ldots$

Table Maker Dilemma:
One cannot decide a priori how many guard digits are reqired to achieve exact rounding for a given combination of transcendent function f and argument x.

IEEE754 guarantees exact rounding for $\oplus, \ominus, \odot, \oslash$, and $\sqrt{x}$.

Further Problems / Properties

The following has to be considered:

- Floating-point arithmetics don't have the associative and distributive properties, i.e., the order of operations matters!
- There is $y \in \mathbb{F}, y \neq 0$, so that $x \oplus y = x$
- Example: $(\epsilon \oplus 1) \ominus 1 = 1 \ominus 1 = 0 \neq \epsilon = \epsilon \oplus 0 = \epsilon \oplus (1 \ominus 1)$
- But the commutative property holds:
  $x \circledast y = y \circledast x$ for $\circledast \in \{\oplus, \odot\}$
- Some further simple rules that are valid:
  - $(-x) \odot y = -(x \odot y)$
  - $1 \odot x = x \oplus 0 = x$
  - $x \odot y = 0 \implies x = 0 \vee y = 0$
  - $x \odot z \leq y \odot z \quad \text{if} \quad x \leq y \wedge z > 0$

Contents I

Error Analysis

Rounding errors are propagated by computations.

- Let $F\colon \mathbb{R}^m \to \mathbb{R}^n$, in components $F(x) = \begin{pmatrix} F_1(x_1,\dots,x_m) \\ \vdots \\ F_n(x_1,\dots,x_m) \end{pmatrix}$
- Compute F in a computer using *numerical realization*
  $F'\colon \mathbb{F}^m \to \mathbb{F}^n$.
  F' is an *algorithm*, i.e., consists of
  - finitely many (= termination)
  - elementary (= known, i.e., $\oplus, \ominus, \odot, \oslash$)

  operations:
  $$F'(x) = \varphi_l(\dots\varphi_2(\varphi_1(x))\dots)$$

Error Analysis

Important:

1. A given F typically has many different realizations, because of different *orders of computation*

$$a + b + c \approx (a \oplus b) \oplus c \neq a \oplus (b \oplus c)!$$

2. Every step φ_i contributes some (unknown) error.
3. In principle, the computational accuracy can be improved arbitrarily, i.e., we have a sequence $(F')^{(k)} \colon (\mathbb{F}^{(k)})^m \to (\mathbb{F}^{(k)})^n$. But in the following we consider only a given fixed finite precision.

Error Analysis

$$F(x) - F'(\mathrm{rd}(x)) = \underbrace{F(x) - F(\mathrm{rd}(x))}_{\text{conditional analysis}} + \underbrace{F(\mathrm{rd}(x)) - F'(\mathrm{rd}(x))}_{\text{rounding error analysis}} \quad (4)$$

Where:

- $F(x)$: exact result
- $F'(\mathrm{rd}(x))$: numerical evaluation
- $F(\mathrm{rd}(x))$: exact result for $\mathrm{rd}(x) \approx x$

From now on:

- "first order" analysis
- absolute / relative errors

Differential Condition Analysis

We assume that $F\colon \mathbb{R}^m \to \mathbb{R}^n$ is twice continuously differentiable.
Taylor's theorem holds for the components F_i:

$$F_i(x + \Delta x) = F_i(x) + \sum_{j=1}^{m} \frac{\partial F_i}{\partial x_j}(x)\Delta x_j + R_i^F(x; \Delta x) \quad i = 1, \dots, n.$$

The remainder is

$$R_i^F(x; \Delta x) = \mathcal{O}\left(\|\Delta x\|^2\right),$$

i.e., the approximation error is quadratic in Δx.

Differential Condition Analysis

Therefore, we can rearrange Taylor's formula:

$$F_i(x + \Delta x) - F_i(x) = \underbrace{\sum_{j=1}^{m} \frac{\partial F_i}{\partial x_j}(x)\Delta x_j}_{\text{leading (first) order}} + \underbrace{R_i^F(x; \Delta x)}_{\text{higher orders}}$$

One often omits higher order terms and writes "$\doteq$" instead of "$=$".

Differential Condition Analysis

Then we have:

$$\frac{F_i(x+\Delta x)-F_i(x)}{F_i(x)} \doteq \sum_{j=1}^{m} \frac{\partial F_i}{\partial x_j}(x)\frac{\Delta x_j}{F_i(x)} \doteq \sum_{j=1}^{m} \underbrace{\left(\frac{\partial F_i}{\partial x_j}(x)\frac{x_j}{F_i(x)}\right)}_{\text{amplification factor } k_{ij}(x)} \cdot \underbrace{\left(\frac{\Delta x_j}{x_j}\right)}_{\leq \text{eps}}, \tag{5}$$

i.e., the amplification factors $k_{ij}(x)$ specify how (relative) input errors $\frac{\Delta x_j}{x_j}$ contribute to (relative) errors in the i-th comp. of F!

Condition

Definition 9 (Condition)

We call the evaluation $y = F(x)$ "ill-conditioned" in point x, iff $|k_{ij}(x)| \gg 1$, else "well-conditioned".
$|k_{ij}(x)| < 1$ is error dampening, $|k_{ij}(x)| > 1$ is error amplification.

The symbol "$\gg$" means "much larger than". Normally this means one number is several orders of magnitude larger than another (e.g., 1 million $\gg$ 1).

This definition is a continuum: there is no sharp separation between "well-conditioned" and "ill-conditioned"!

Example I

Example 10

1. Addition: $F(x_1, x_2) = x_1 + x_2$, $\frac{\partial F}{\partial x_1} = \frac{\partial F}{\partial x_2} = 1$.
   According to our formula:

$$\frac{F(x_1 + \Delta x_1, x_2 + \Delta x_2) - F(x_1, x_2)}{F(x_1, x_2)}$$

$$\doteq \underbrace{1 \cdot \frac{x_1}{x_1 + x_2}}_{=k_1} \frac{\Delta x_1}{x_1} + \underbrace{1 \cdot \frac{x_2}{x_1 + x_2}}_{=k_2} \frac{\Delta x_2}{x_2}$$

   Ill-conditioned for $x_1 \to -x_2$!

Example II

Example 10

2. $F(x_1, x_2) = x_1^2 - x_2^2, \frac{\partial F}{\partial x_1} = 2x_1, \frac{\partial F}{\partial x_2} = -2x_2.$

$$\frac{F(x_1 + \Delta x_1, x_2 + \Delta x_2) - F(x_1, x_2)}{F(x_1, x_2)}$$

$$\doteq \underbrace{2x_1 \cdot \frac{x_1}{x_1^2 - x_2^2}}_{=k_1} \frac{\Delta x_1}{x_1} + \underbrace{(-2x_2) \cdot \frac{x_2}{x_1^2 - x_2^2}}_{=k_2} \frac{\Delta x_2}{x_2}$$

$$\implies k_1 = \frac{2x_1^2}{x_1^2 - x_2^2}, k_2 = -\frac{2x_2^2}{x_1^2 - x_2^2},$$

Ill-conditioned for $|x_1| \approx |x_2|$.

Contents I

Rounding Error Analysis

Also known as "forward rounding error analysis", there are other variants.

After error decomposition, Eq. (4):
consider $F(x) - F'(x)$ with $x \in \mathbb{F}^m$, F' "composed" from single operations $\circledast \in \{\oplus, \ominus, \odot, \oslash\}$

Eq. (3) (exactly rounded arithmetics) and Lemma 7 (rounding error) imply

$$\frac{(x \circledast y) - (x * y)}{(x * y)} = \epsilon \quad \text{with } |\epsilon| \leq \text{eps}$$

Careful, ϵ depends on x and y, and therefore is different for each individual operation!

$$\implies x \circledast y = (x * y) \cdot (1 + \epsilon) \quad \text{for an } |\epsilon(x, y)| \leq \text{eps}$$

Example I

Example 11

$F(x_1, x_2) = x_1^2 - x_2^2$ with two different realizations:

1. $F_a(x_1, x_2) = (x_1 \odot x_1) \ominus (x_2 \odot x_2)$
2. $F_b(x_1, x_2) = (x_1 \ominus x_2) \odot (x_1 \oplus x_2)$

First realization:

$$u = x_1 \odot x_1 = (x_1 \cdot x_1) \cdot (1 + \epsilon_1)$$
$$v = x_2 \odot x_2 = (x_2 \cdot x_2) \cdot (1 + \epsilon_2)$$
$$F_a(x_1, x_2) = u \ominus v = (u - v) \cdot (1 + \epsilon_3)$$
$$\frac{F_a(x_1, x_2) - F(x_1, x_2)}{F(x_1, x_2)} \doteq \frac{x_1^2}{x_1^2 - x_2^2}(\epsilon_1 + \epsilon_3) + \frac{x_2^2}{x_2^2 - x_1^2}(\epsilon_2 + \epsilon_3)$$

Example II

Example 11

Second realization:

$$u = x_1 \ominus x_2 = (x_1 - x_2) \cdot (1 + \epsilon_1)$$
$$v = x_1 \oplus x_2 = (x_1 + x_2) \cdot (1 + \epsilon_2)$$
$$F_b(x_1, x_2) = u \odot v = (u \cdot v) \cdot (1 + \epsilon_3)$$
$$\frac{F_b(x_1, x_2) - F(x_1, x_2)}{F(x_1, x_2)} \doteq \frac{x_1^2 - x_2^2}{x_1^2 - x_2^2}(\epsilon_1 + \epsilon_2 + \epsilon_3) = \epsilon_1 + \epsilon_2 + \epsilon_3$$

$\Longrightarrow$ second realization is better than first realization.

Numerical Stability

Definition 12 (Numerical Stability)

We call a numerical algorithm “numerically stable”, if the rounding errors accumulated during computation have the same order of magnitude as the unavoidable problem error from condition analysis.

In other words:
Amplification factors from rounding analysis $\leq$ those from condition analysis $\implies$ “numerically stable”

Both realizations a, b from Ex. 11 are numerically stable.

Quadratic Equation

Let $p^2/4 > q \neq 0$, then the equation

$$y^2 - py + q = 0$$

has two real and separate solutions

$$y_{1,2} = f_{\pm}(p, q) = \frac{p}{2} \pm \sqrt{\frac{p^2}{4} - q}. \quad \text{(defines two } f\text{!)}$$

Condition analysis with $D := \sqrt{\frac{p^2}{4} - q}$:

$$\frac{f(p + \Delta p, q + \Delta q) - f(p, q)}{f(p, q)}$$

$$\doteq \left(1 \pm \frac{p}{2D}\right) \frac{p}{p \pm 2D} \frac{\Delta p}{p} - \frac{q}{D\,(p \pm 2D)} \frac{\Delta q}{q}$$

Quadratic Equation

This means:

- For $\frac{p^2}{4} \gg q$ and $p < 0$

$$f_-(p,q) = \frac{p}{2} - \sqrt{\frac{p^2}{4} - q}$$

is well-conditioned.

- For $\frac{p^2}{4} \gg q$ and $p > 0$

$$f_+(p,q) = \frac{p}{2} + \sqrt{\frac{p^2}{4} - q}$$

is well-conditioned.

- For $\frac{p^2}{4} \approx q$ both f_+ and f_- are ill-conditioned, this cannot be avoided.

Quadratic Equation

Numerically handy evaluation for the case $\frac{p^2}{4} \gg q$:

$p < 0$:
Compute $y_2 = \frac{p}{2} - \sqrt{\frac{p^2}{4} - q}$, then $y_1 = \frac{q}{y_2}$ using Vieta's Theorem ($q = y_1 \cdot y_2$).

$p > 0$:
Compute $y_1 = \frac{p}{2} + \sqrt{\frac{p^2}{4} - q}$, then $y_2 = \frac{q}{y_1}$.

$\Longrightarrow$ every problem has to be considered individually!

Cancellation

The discussed examples contain the phenomenon of *cancellation*.
It appears during

- addition $x_1 + x_2$ with $x_1 \approx -x_2$
- subtraction $x_1 - x_2$ with $x_1 \approx x_2$

Remark 13

Cancellation means extreme amplification of errors introduced *before* the addition or subtraction.

If $x_1, x_2 \in \mathbb{F}$ are *machine numbers*, then

$$\left| \frac{(x_1 \ominus x_2) - (x_1 - x_2)}{(x_1 - x_2)} \right| \leq \text{eps}$$

holds, so this is not problematic. The problem of cancellation only occurs if x_1 and x_2 already contain errors.

Example

Example 14

Consider $\mathbb{F} = \mathbb{F}(10, 4, 1)$.

$x_1 = 0.11258762 \cdot 10^2, x_2 = 0.11244891 \cdot 10^2$
$\implies \mathrm{rd}(x_1) = 0.1126 \cdot 10^2, \mathrm{rd}(x_2) = 0.1124 \cdot 10^2$

$x_1 - x_2 = 0.13871 \cdot 10^{-1}$, but $\mathrm{rd}(x_1) - \mathrm{rd}(x_2) = 0.2 \cdot 10^{-1}$

The result has not a single valid digit! Relative error:

$$\frac{0.2 \cdot 10^{-1} - 0.13871 \cdot 10^{-1}}{0.13871 \cdot 10^{-1}} \approx 0.44 \approx 883 \cdot \underbrace{\frac{1}{2} \cdot 10^{-3}}_{=\mathrm{eps}}!$$

Basic Rule

In the given example: error caused by rounding of arguments.

Source of errors is irrelevant, this also happens if x_1, x_2 contain errors from previous computation steps.

Rule 15
Employ potentially dangerous operations as soon as possible in algorithms, when the least possible amount of errors has been accumulated (compare Ex. 11).

Exponential Function

The function $\exp(x) = e^x$ can be written as a power series for all $x \in \mathbb{R}$:

$$\exp(x) = \sum_{k=0}^{\infty} \frac{x^k}{k!}$$

Obvious approach: truncate calculation after n terms, $\exp(x) \approx \sum_{k=0}^{n} \frac{x^k}{k!}$.

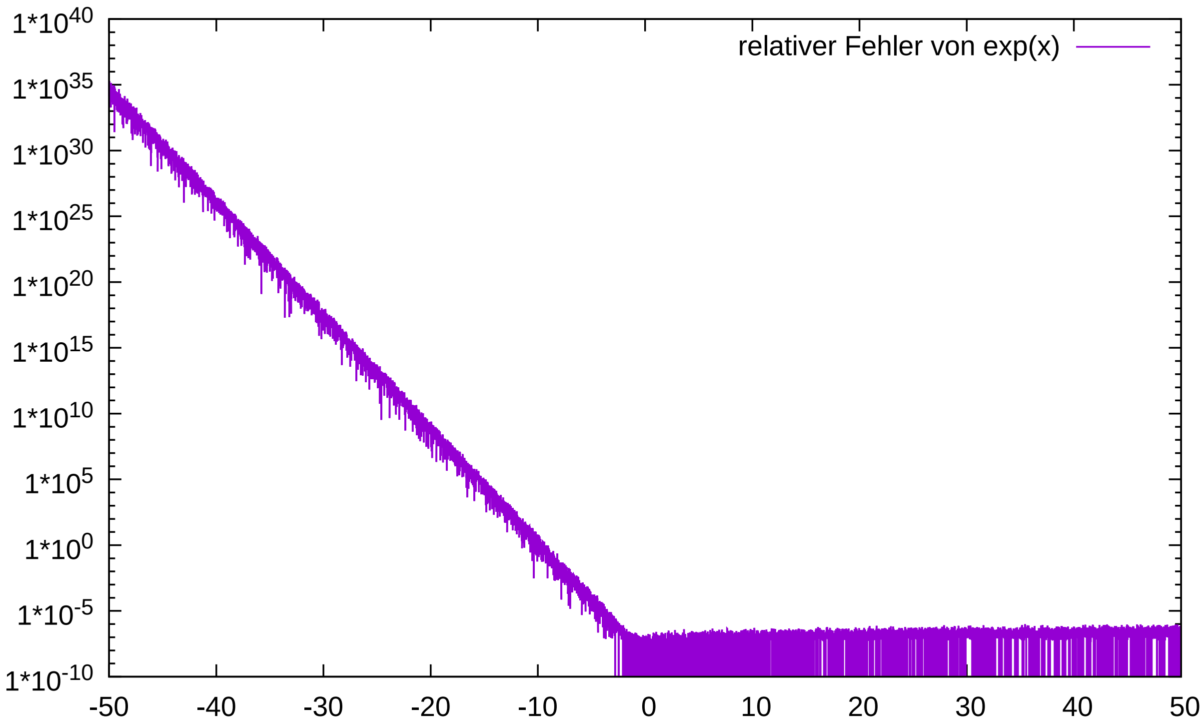
Use recursion:

$$\begin{aligned} & y_0 := 1, \quad S_0 := y_0 = 1, \\ \forall k > 0: \quad & y_k := \frac{x}{k} \cdot y_{k-1}, \quad S_k := S_{k-1} + y_k \end{aligned}$$

y_n: terms of series, S_n: partial sums

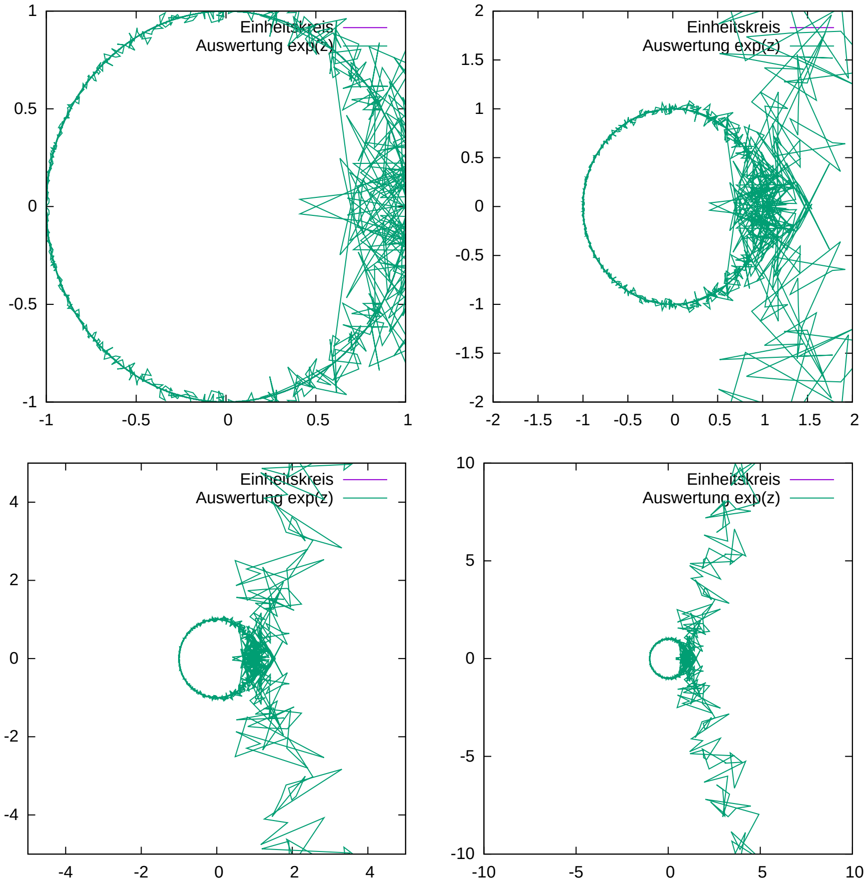
Error for Different Values of x

Results for recursion formula with **`float`**, $n = 100$:



- Negative values of x lead to arbitrarily large errors
- This effect is *not* caused by the truncation of the series!

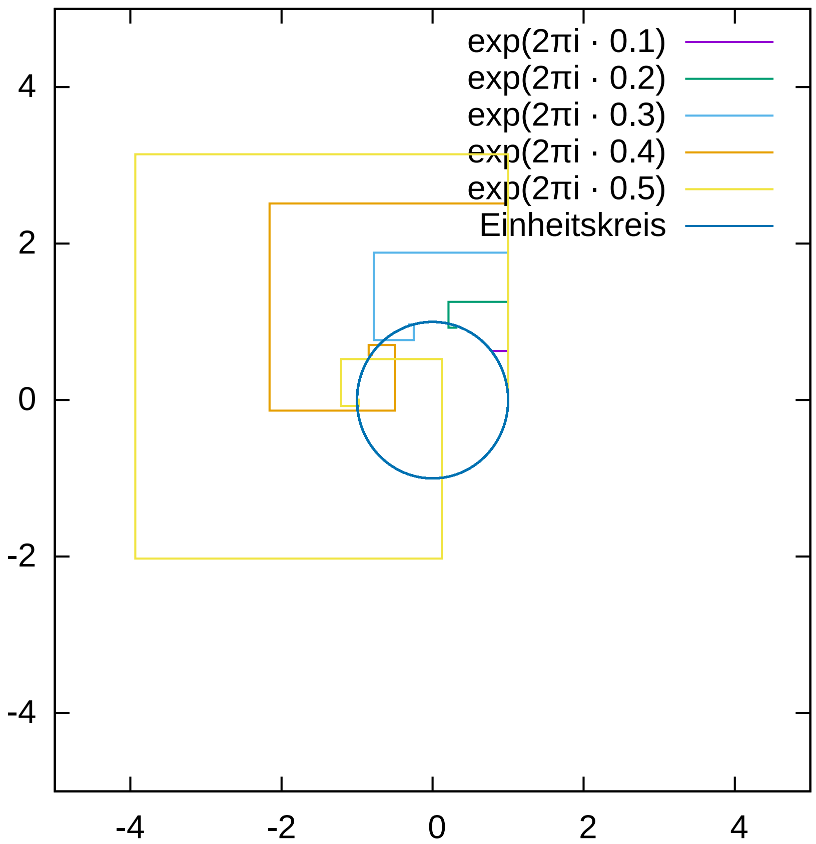
Deviations for Imaginary Arguments



Results for the imaginary interval $[-50, 50] \cdot i$

- For $|z| \leq \pi$ the result is somewhat acceptable
- For $|z| \to 2\pi$ the error continues to grow
- Then the values leave the circle (the trajectory approaches a straight line and won't return)

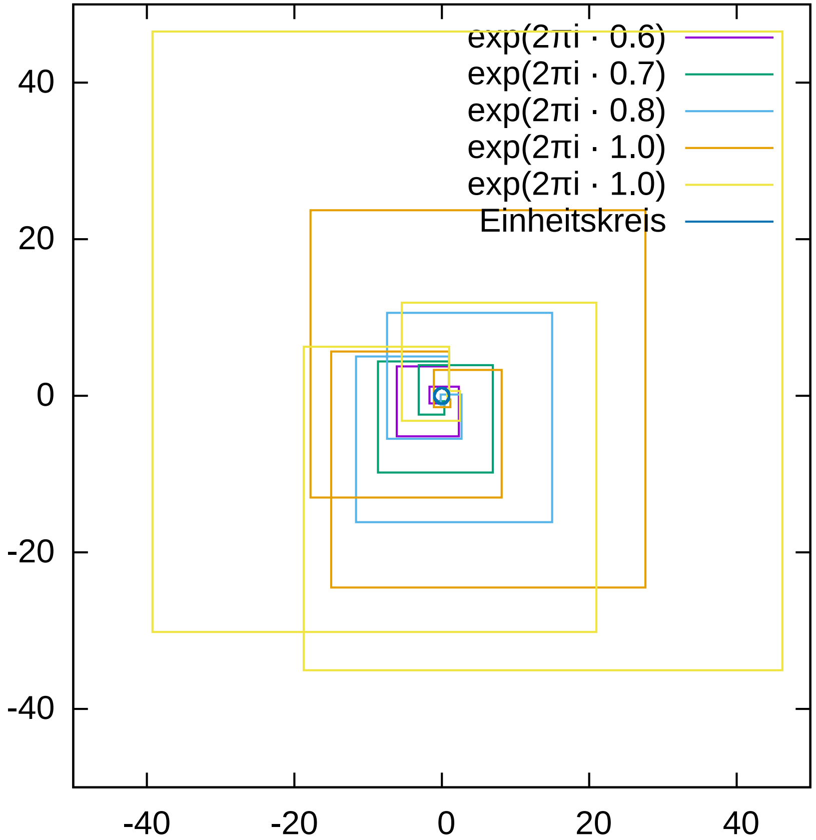
Visualization of Convergence Behavior



- even powers contribute to the real part of $\exp(2\pi i \cdot x)$
- odd powers contribute to the imaginary part

$\Longrightarrow$ addition of terms alternates between changes to real and imaginary part

Visualization of Convergence Behavior



- Absolute value of intermediate results grows exponentially in x
- Shape of trajectory looks more and more like a square

$\Longrightarrow$ cancellation

Condition Analysis for $\exp(x)$

For the function exp we have $\exp' = \exp$, and therefore:

$$\frac{\exp(x+\Delta x)-\exp(x)}{\exp(x)} \doteq \left(\exp'(x)\frac{x}{\exp(x)}\right)\cdot\left(\frac{\Delta x}{x}\right) = \Delta x$$

$\implies$ absolute error of x becomes relative error of $\exp(x)$
(compare: exp is isomorphism between $(\mathbb{R},+)$ and $(\mathbb{R}_+,\cdot)$.)

$k = x$ means exp is well-conditioned if x is not too large
$\implies$ considered algorithm is unstable for $x < 0$

Is there a more stable algorithm? $\rightsquigarrow$ exercise

Recursion Formula for Integrals

Integrals of the form

$$I_k = \int_0^1 x^k \exp(x)\, dx$$

can be solved using a recursion formula:

$$I_0 = e - 1, \quad \forall k > 0\colon I_k = e - k \cdot I_{k-1}$$

We have a primitive integral for the first term in the sequence, because $\exp'(x) = \exp(x)$, other terms can be computed using the formula above.

How well does this work in practice?

Recursion Formula for Integrals

The first 26 values of $\{I_k\}_k$, computed with finite precision:

k	computed I_k	error $\|\Delta I_k\|$	k	computed I_k	error $\|\Delta I_k\|$
0	1.718281828459050	$2.6 \cdot 10^{-15}$	13	0.181983054536145	$3.3 \cdot 10^{-7}$
1	1	(zero)	14	0.170519064953013	$4.6 \cdot 10^{-6}$
2	0.718281828459045	$1.5 \cdot 10^{-15}$	15	0.160495854163853	$7.0 \cdot 10^{-5}$
3	0.563436343081910	$5.5 \cdot 10^{-16}$	16	0.150348161837404	$1.1 \cdot 10^{-3}$
4	0.464536456131406	$1.0 \cdot 10^{-15}$	17	0.162363077223183	$1.9 \cdot 10^{-2}$
5	0.395599547802016	$6.0 \cdot 10^{-15}$	18	-0.204253561558257	$3.4 \cdot 10^{-1}$
6	0.344684541646949	$3.8 \cdot 10^{-14}$	19	6.59909949806592	$6.7 \cdot 10^{0}$
7	0.305490036930402	$2.7 \cdot 10^{-13}$	20	-129.263708132859	$1.3 \cdot 10^{1}$
8	0.274361533015832	$2.1 \cdot 10^{-12}$	21	2717.25615261851	$2.7 \cdot 10^{3}$
9	0.249028031316559	$1.9 \cdot 10^{-11}$	22	-59776.9170757787	$6.0 \cdot 10^{4}$
10	0.228001515293454	$1.9 \cdot 10^{-10}$	23	1374871.81102474	$1.4 \cdot 10^{6}$
11	0.210265160231056	$2.1 \cdot 10^{-9}$	24	-32996920.7463119	$3.3 \cdot 10^{7}$
12	0.195099905686377	$2.5 \cdot 10^{-8}$	25	824923021.376079	$8.2 \cdot 10^{8}$

Recursion formula $I_k = e - k \cdot I_{k-1}$ leads to error amplification by a factor of k in k-th step!

Better Options

1. All I_k are of the form $a \cdot e + b$, where $a, b \in \mathbb{Z}$. Compute these numbers using the recursion formula, and use floating-point numbers only in the last step of computation.
2. Flip the recursion formula: if $I_k \to I_{k+1}$ amplifies the error by k, then $I_{k+1} \to I_k$ reduces it by k!

Because of $0 \leq x^k \leq 1$ and $0 \leq \exp(x) \leq 3$ on $[0, 1]$, $0 \leq I_k \leq 3$ must hold. If we more or less arbitrarily set, e.g., $I_{50} := 1.5$, then the error can be at most 1.5.

Use inverted recursion formula

$$I_k = (k+1)^{-1} \cdot (e - I_{k+1}).$$

Recursion Formula for Integrals

The values for I_k between $k = 25$ and 50, calculated backwards:

k	computed I_k	error $\|\Delta I_k\|$	k	berechnetes I_k	Fehler $\|\Delta I_k\|$
50	1.5	$1.4 \cdot 10^{0}$	37	0.0697442966294832	$2.8 \cdot 10^{-16}$
49	0.0243656365691809	$2.9 \cdot 10^{-2}$	36	0.0715820954548530	$1.9 \cdot 10^{-16}$
48	0.0549778814671401	$5.9 \cdot 10^{-4}$	35	0.0735194370278942	$2.8 \cdot 10^{-17}$
47	0.0554854988956647	$1.2 \cdot 10^{-5}$	34	0.0755646397551757	$2.2 \cdot 10^{-16}$
46	0.0566552410545400	$2.6 \cdot 10^{-7}$	33	0.0777269761383491	$2.7 \cdot 10^{-18}$
45	0.0578614475522718	$5.7 \cdot 10^{-9}$	32	0.0800168137066878	$1.8 \cdot 10^{-16}$
44	0.0591204529090394	$1.3 \cdot 10^{-10}$	31	0.0824457817110112	$2.6 \cdot 10^{-16}$
43	0.0604354858079547	$2.9 \cdot 10^{-12}$	30	0.0850269692499366	$2.9 \cdot 10^{-16}$
42	0.0618103800616533	$6.7 \cdot 10^{-14}$	29	0.0877751619736370	$4.5 \cdot 10^{-17}$
41	0.0632493201999379	$1.8 \cdot 10^{-15}$	28	0.0907071264305313	$4.3 \cdot 10^{-16}$
40	0.0647568904453441	$1.4 \cdot 10^{-16}$	27	0.0938419536438755	$4.7 \cdot 10^{-16}$
39	0.0663381234503425	$2.1 \cdot 10^{-16}$	26	0.0972014768450063	$1.3 \cdot 10^{-17}$
38	0.0679985565386847	$1.5 \cdot 10^{-16}$	25	0.1008107827543860	$1.1 \cdot 10^{-16}$

Despite a completely unusable estimate for the initial value I_{50}, the new recursion formula $I_k = (k+1)^{-1} \cdot (e - I_{k+1})$ quickly leads to very good results!

Idea of Error Estimates

Error analysis for initial value $I_{k+m}, m \geq 1$:

$$|\Delta I_k| \approx \frac{k!}{(k+m)!} |\Delta I_{k+m}| \leq \frac{k!}{(k+m)!} \cdot 1.5 \leq (k+1)^{-m} \cdot 1.5$$

Idea: compute required number of steps m from desired accuracy $|\Delta I_k| <$ tol.

$$(k+1)^{-m} \cdot 1.5 < \text{tol} \implies \exp(-m \cdot \ln(k+1)) < \frac{\text{tol}}{1.5}$$
$$\implies -m \cdot \ln(k+1) < \ln\left(\frac{\text{tol}}{1.5}\right) \implies m > \left|\frac{\ln(\text{tol}) - \ln(1.5)}{\ln(k+1)}\right|$$

Example: $k = 25$, $\text{tol} = 10^{-8} \implies m > 5.7$

Idea of Error Estimates

Result for $m = 6$:

k	computed I_k	error $\lvert\Delta I_k\rvert$
31	1.5	$1.4 \cdot 10^{0}$
30	0.0392994138212595	$4.6 \cdot 10^{-2}$
29	0.0892994138212595	$1.5 \cdot 10^{-3}$
28	0.0906545660219926	$5.3 \cdot 10^{-5}$
27	0.0938438308013233	$1.9 \cdot 10^{-6}$
26	0.0972014073206564	$7.0 \cdot 10^{-8}$
25	0.1008107854284000	$2.9 \cdot 10^{-9}$

- Inverted recursion formula is numerically stable, in contrast to naive approach
- Error estimate minimizes effort for prescribed accuracy

$\Longrightarrow$ stable and efficient