Floating-Point Numbers

Pitfalls with IEEE-754

Laurent Hoeltgen

June 23rd, 2022

Why All This?

Floating point computations on computers are tricky!

> ‘ Even with 10^7 iterations, the residual of my linear system never drops below 10^{-16}. (former student of mine) ’

> ‘ [...] if you were as naive as me and thought computer numbers would work just like real numbers. (former Bosch colleague) ’

It Can't Be That Bad, Can It?[1]

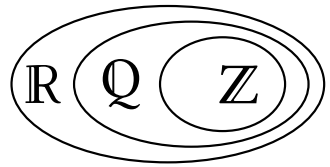
- Vancouver Stock Exchange, 1982, accumulated wrong rounding caused certain indices to lose 50% of their value.
- Patriot Missile Failure, 1991, 128 deaths due to a rounding error in the conversion of float to double for time stamps.
- Parliamentary elections in Schleswig-Holstein, 1992, a bug in Excel causes a undesired rounding and resulted in the green party reaching the 5% threshold.
- Ariane 5, flight 501, 1996, 500M dollar in damages due to inaccurate unit conversions.

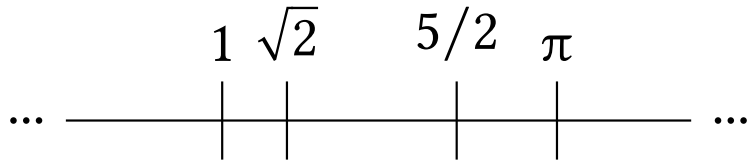
[1] *Hidden Danger: The Problem* 2022.

Overview

What's a Number Anyway?[2]

$$\mathbb{N} \subset \mathbb{Z} \subset \mathbb{Q} \subset \mathbb{R} \subset \mathbb{C}$$





- $\mathbb{N}$: $\{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}, ...\}$ (Zermelo-Fraenkel)
- $\mathbb{Z}$: Equivalence relation on $\mathbb{N} \times \mathbb{N}$
- $\mathbb{Q}$: Equivalence relation on $\mathbb{Z} \times \mathbb{Z} \setminus \{0\}$
- $\mathbb{R}$: Equivalence relation of pairs of sequences from $\mathbb{Q}$
- $\mathbb{C}$: Quotient space $\mathbb{R}[x]/(x^2 + 1)$

Also: $\mathbb{F}_q$, $\mathbb{Q}(\sqrt{2})$, $\mathbb{H}$, ...

[2] Wikipedia 2020; Bedürftig and Murawski 2019.

1. Μονάς ἐστιν, καθ' ἣν ἕκαστον τῶν ὄντων ἓν λέγεται
   A unit is that by virtue of which each of the things that exist is called one.
2. Ἀριθμὸς δὲ τὸ ἐκ μονάδων συγκείμενον πλῆθος.
   A number is a multitude composed of units.

(Euklid, Elements VII, Defintion 1 & 2)

Number Fields: Definition

A set G with a mapping $\circ$ is a *commutative group* if

1. G is algebraically closed with respect to $\circ$.
2. $x \circ y = y \circ x$ and $(x \circ y) \circ z = x \circ (y \circ z), \quad \forall x, y, z \in \mathrm{G}$
3. There exists a *neutral* and *inverse* element for every $x \in \mathrm{G}$.

A set $\mathbb{K}$ with two mappings $\oplus$ und $\odot$ is a *number field* if

1. $(\mathbb{K}, \oplus)$ is a *commutative group* with neutral element $e_\oplus$.
2. $(\mathbb{K} \setminus \{e_\oplus\}, \odot)$ is a *commutative group* with neutral element $e_\odot$.
3. The following distributive rules hold

$$a \odot (b \oplus c) = (a \odot b) \oplus (a \odot c), \quad \forall a, b, c \in \mathbb{K}$$
$$(a \oplus b) \odot c = (a \odot c) \oplus (b \odot c), \quad \forall a, b, c \in \mathbb{K}$$

Real vs. Floating-Point Numbers[3]

Property	$\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$	Floating-Point
Closedness	✓	✓ / ✗
Commutative	✓	✓
Associative	✓	✗
Unit element	✓	✓
Inverse element	✓	✗
Distributivity	✓	✗
Monotonicity	✓	✓ / ✗

[3] Shepherd 2016; Goldberg 1991.

Loss of Mathematical Properties I[4]

- *Closedness* under + and ·, if $\pm\infty$ and NaN are allowed
- *Monotonicity* is (almost) given

$$x \geqslant y \implies x + z \geqslant y + z, \quad \forall x, y, z$$
$$x \geqslant y \implies x \cdot z \geqslant y \cdot z, \quad \forall x, y, \ \forall z \geqslant 0$$

  Exception: $\pm\infty$ and NaN
- *Commutativity* ist granted for + and ·

[4] Nicholas John Higham 2002.

Loss of Mathematical Properties II[5]

- Real Numbers are *associative*

$$x + (y + z) = (x + y) + z, \quad \forall x, y, z \in \mathbb{R}$$

- Floating-point numbers are not assiciative

```
0.1 + (0.2 + 0.3) # 0.6
(0.1 + 0.2) + 0.3 # 0.6000000000000001
```

Listing: No associativity of floats (Python/Ruby)

[5] Nicholas John Higham 2002; The Python Project 2020; The Ruby Project 2018.

Loss of Mathematical Properties III[6]

- For $+$ there is *always* an *inverse element*
- For $\cdot$ this is not true

  10.0 is a floating point number, 0.1 is *not*

  $[0.1]_{10}$ in binary is $[0.\overline{00011}]_2$.
- x/y is *not* the same as $x \cdot (1/y)$.

```
1 program inverse
2     use iso_fortran_env, only: real64, output_unit
3     implicit none
4     real(real64)::t = 3.1416D0
5     real(real64)::x = 651370000000.0D0
6     real(real64)::y = 1.0D0/3.1416D0
7     write(output_unit,'(F17.4)') (x / t) * t !! 651370000000.0000
8     write(output_unit,'(F17.4)') (x * y) * t !! 651370000000.0001
9 end program inverse
```

Listing: Issue with inverses (Fortran)

[6] Edelman 2020; Lemire 2019; The GCC developers 2019.

Loss of Mathematical Properties IV[7]

Real numbers are *distributive*

$$x \cdot (y + z) = x \cdot y + x \cdot z, \quad \forall x, y, z \in \mathbb{R}$$

```
function distributive()
    x::Float32 = 1.0e20
    println("Version 1:", x*(x-x)) # 0.0
    println("Version 2:", x*x-x*x) # NaN
end
```

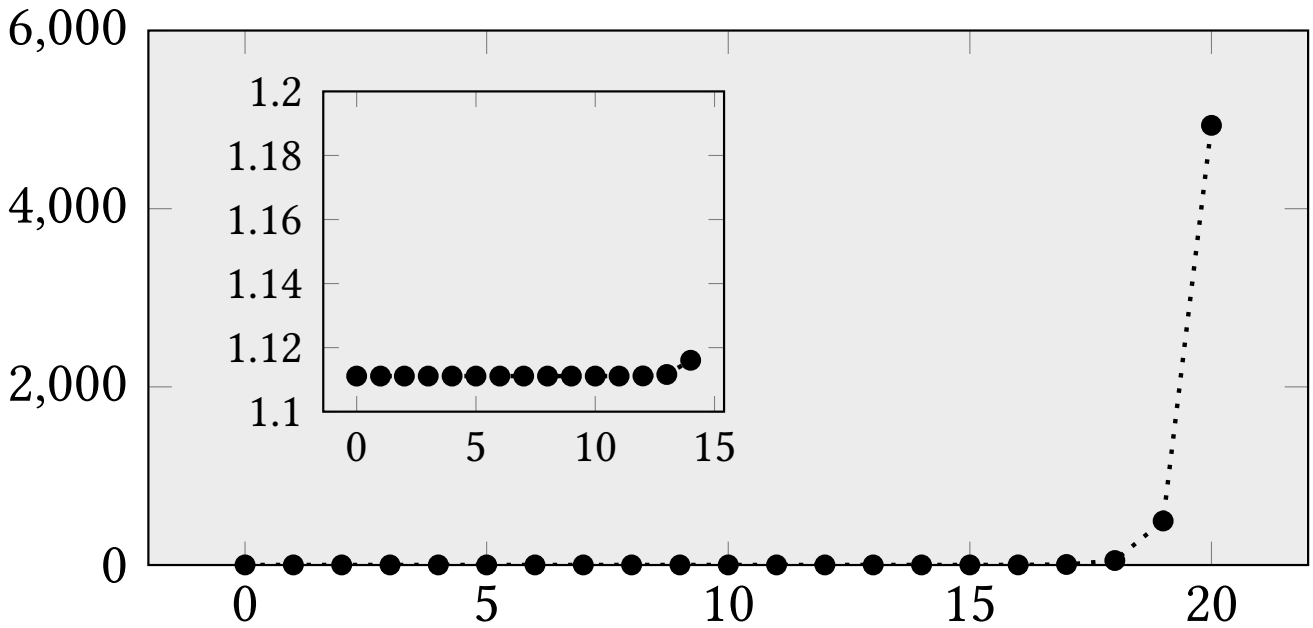
Listing: No distributivity for floating point numbers (Julia)

[7] Shepherd 2016; The Julia Project 2019.

Accumulation of Inaccuracies[8]

Consider the following sequence.

$$\begin{cases} u_0 & = \frac{10}{9} = 1.\bar{1} \\ u_{k+1} & = (u_k - 1) * 10 = u_0 \end{cases}$$



In each iteration we lose one decimal in precision.

[8] PseudoRandom 2020.

Deceptive Convergence[9]

$$\begin{cases} u_0 &= 2, \\ u_1 &= -4, \\ u_n &= 111 - \frac{1130}{u_{n-1}} + \frac{3000}{u_{n-1}u_{n-2}} \end{cases}$$

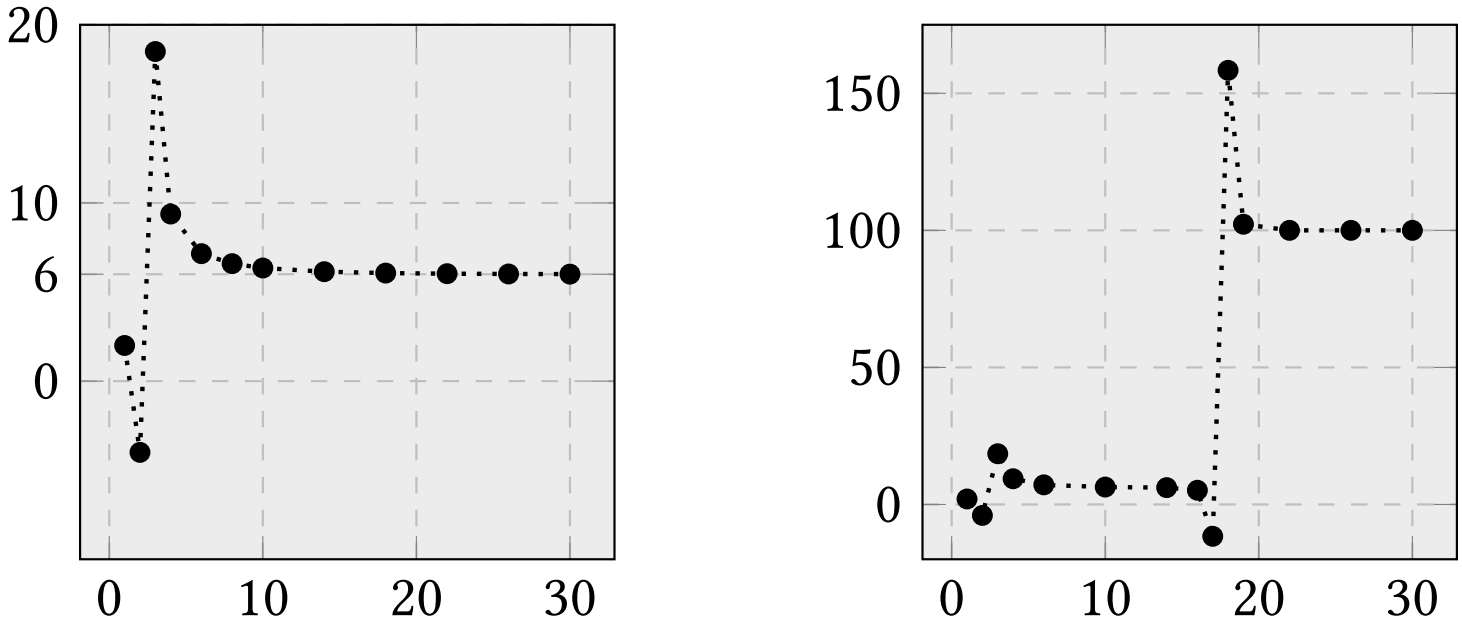


Figure: *Left:* Exact (stops at 6), *Right:* Floating-Point (stops at 100)

[9] Muller 1989.

Fun With Excel[10]

Input	Result
$V = 4/3$	1.3333333333333000E+00
$W = V - 1$	3.3333333333333000E-01
$X = W * 3$	1.0000000000000000E+00
$Y = X - 1$	0.0000000000000000E+00
$Z = Y * 2^{52}$	0.0000000000000000E+00
$(4/3 - 1) * 3 - 1$	0.0000000000000000E+00
$((4/3 - 1) * 3 - 1)$	-2.2204460492503100E-16
$((4/3 - 1) * 3 - 1) * 2^{52}$	-1.0000000000000000E+00

Table: Inconsistencies in the Floating Point Representation

Works with any Excel version since Excel 2000.

[10] Kahan 2006.

Comparison with Fortran

```
1  program ExcelComparison
2      use iso_fortran_env, only: real64, output_unit
3      implicit none
4      real(REAL64) :: p, q, r, v, w, x, y, z
5      v = 4.0D0/3.0D0
6      w = v - 1.0D0
7      x = w * 3.0D0
8      y = x - 1.0D0
9      z = y * 2.0D0**52
10     write(output_unit,'(F20.17)') v !!  1.3333333333333326
11     write(output_unit,'(F20.17)') w !!  0.3333333333333326
12     write(output_unit,'(F20.17)') x !!  0.9999999999999978
13     write(output_unit,'(F20.17)') y !! -0.0000000000000022
14     write(output_unit,'(F20.17)') z !! -1.0000000000000000
15 end program ExcelComparison
```

Listing: Correct Results

Floating-Point Model[11]

$$y = \pm m \cdot \beta^{e-t}, \quad e \in [e_{\min}, e_{\max}]$$
$$= \pm \beta^e \left(\frac{d_1}{\beta} + \frac{d_2}{\beta^2} + ... + \frac{d_t}{\beta^t} \right), \quad d_j \in \{0, ..., \beta - 1\} \; \forall j$$

- $\beta \in \mathbb{N} \setminus \{0, 1\}$ is called *Base*
- $t \in \mathbb{Z}$ is called *Precision*
- $e \in [e_{\min}, e_{\max}]$ is called *Exponent*
- $m \in \{0, ..., \beta^t - 1\} \cap \mathbb{N}$, (resp. $\{\beta^{t-1}, ..., \beta^t - 1\}$) is called *Mantissa*

m und e are properties of the number
β und t are properties of the model

[11] Nicholas John Higham 2002.

Properties of the System[12]

- $m \geqslant \beta^{t-1}$ guarantees uniqueness of the representation
- smallest positive number: $\beta^{e_{\min}-1}$
- largest positive number: $\beta^{e_{\max}}(1-\beta^{-t})$
- *Machine epsilon* (distance between 1 and next number):

$$\varepsilon_M := \beta^{1-t}$$

- *Unit in last place* (ulp): $\beta^e \cdot .00\ldots01 = \beta^{e-t}$.
- *Denormalized* floating-point number:

$$y = \pm m \cdot \beta^{e_{\min}-t}, \quad m \in \{0, \ldots, \beta^{t-1}-1\}$$

less precision, but represent some kind of *safety net*

[12] Nicholas John Higham 2002.

Floating-Point Numbers In The Last Century[13]

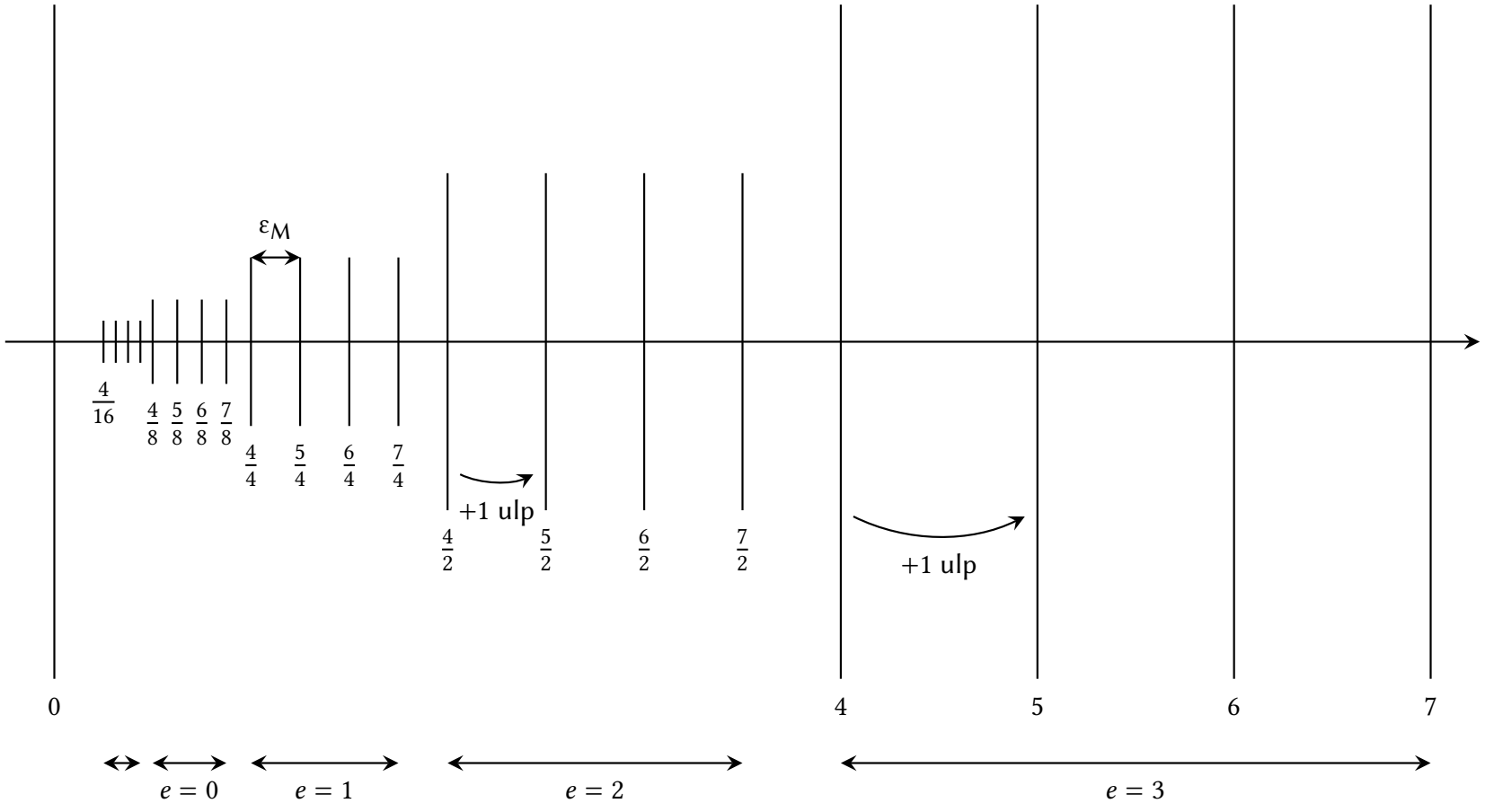
$$y = \pm m \cdot \beta^{e-t} \quad e \in [e_{\min}, e_{\max}]$$

Machine, Arithmetic	β	t	$e_{\min}$	$e_{\max}$
Cray 1, single	2	48	-8192	8191
Cray 1, double	2	96	-8192	8191
DEC VAX G, double	2	53	-1023	1023
DEC VAX D, double	2	56	-127	127
Burroughs 6700, single	8	13	-51	76
HP 28 und HP 48G	10	12	-499	499
IBM 3090 single	16	6	-64	63
IBM 3090 double	16	14	-64	63
Setun und Setun-70	3	-	-	-

[13] Beebe 2017; *Development of ternary computers at Moscow State University* 2020.

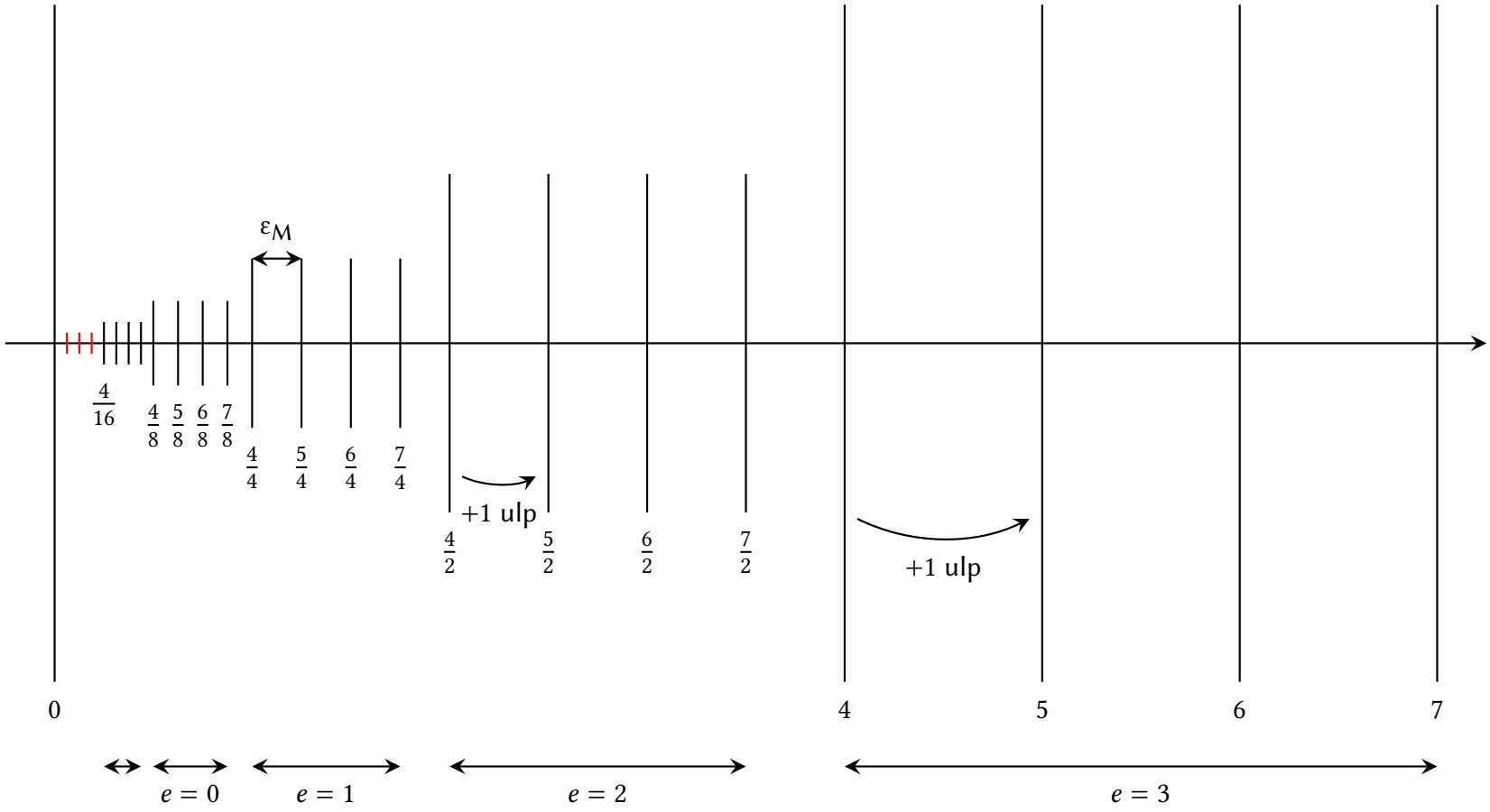
Example

Wir wählen $\beta = 2, t = 3, e_{\text{min}} = -1, e_{\text{max}} = 3$.



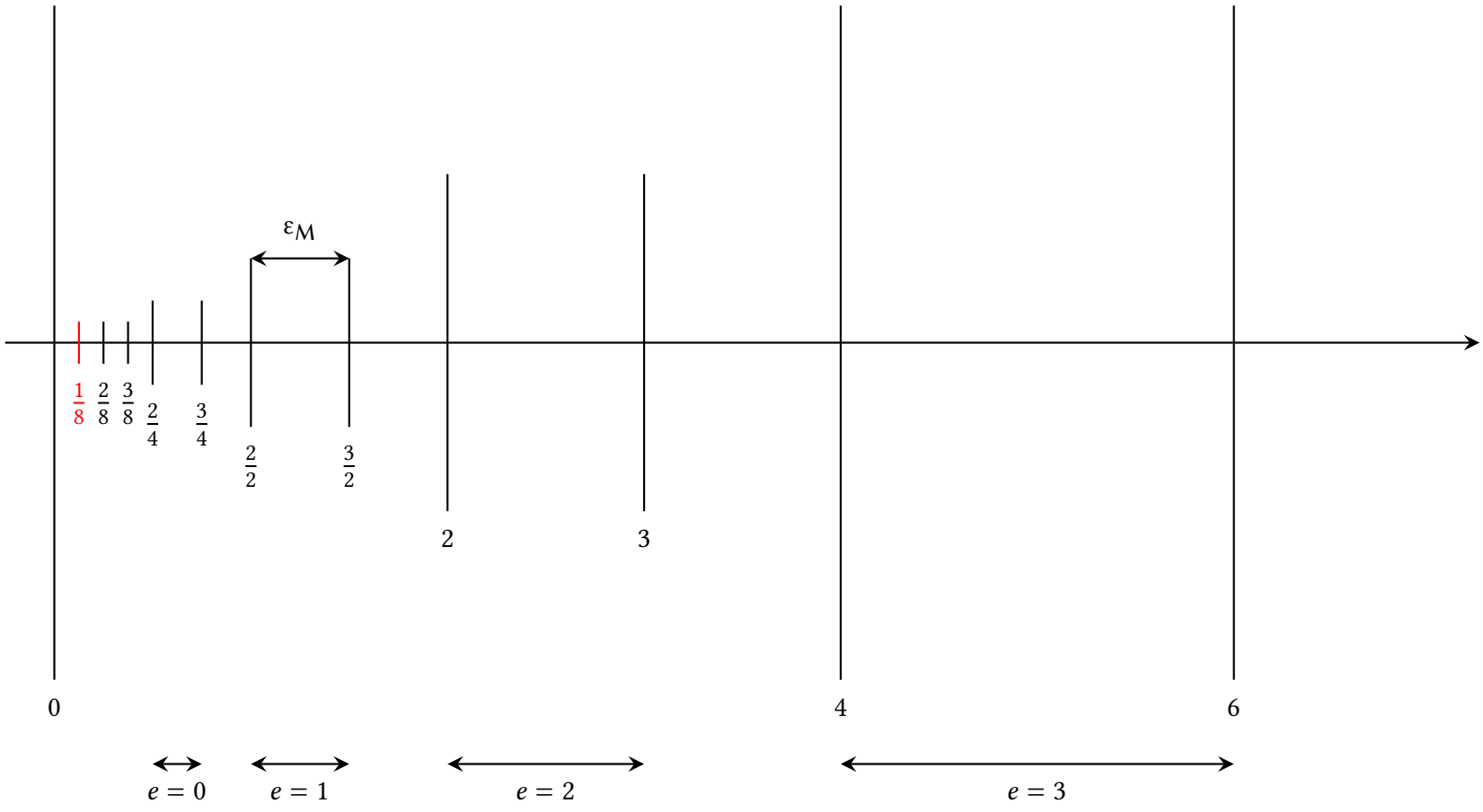
Example

Wir wählen $\beta = 2, t = 3, e_{\text{min}} = -1, e_{\text{max}} = 3$.



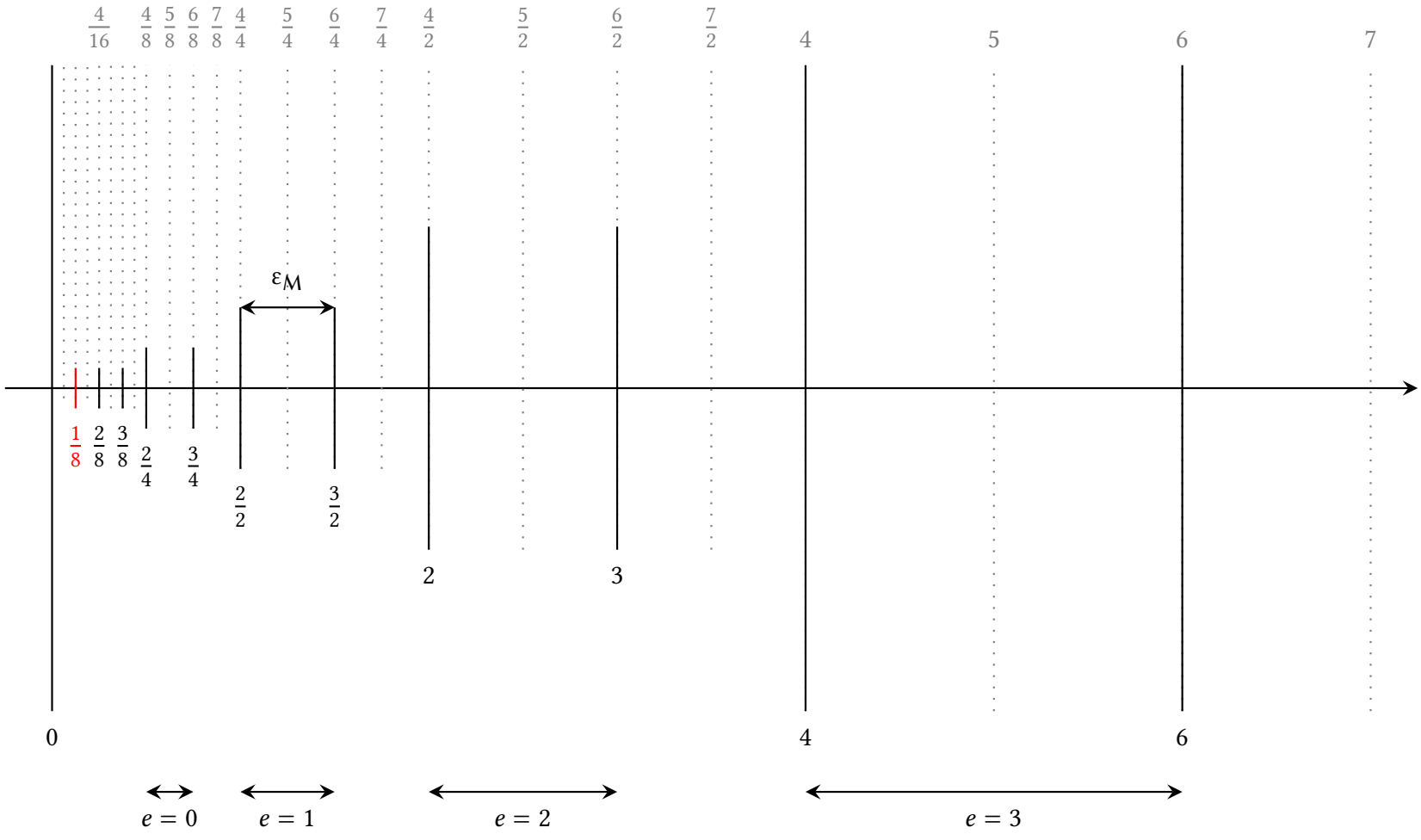
Example

If we chose $t = 2$:



Example

If we chose $t = 2$:



Why Have Denormalized Numbers?[14]

Our floating-point system:

$$\left\{\frac{2}{8}, \frac{3}{8}, \frac{4}{8}, \frac{6}{8}, \frac{8}{8}, \frac{12}{8}, \frac{16}{8}, \frac{24}{8}, \frac{32}{8}, \frac{48}{8}\right\}$$

Without denormalized numbers:

$$\frac{3}{8} - \frac{2}{8} = \frac{1}{8} \to 0$$

Algorithmus 1: Example for problematic code

if $(x < y)$ **then**
  $z \leftarrow \frac{1}{y-x}$
end if

[14] Moler 2014.

The IEEE 754-2008 (ISO/IEC/IEEE 60559:2011) Standard[16]

- Specifies the internal representation of floating point numbers
- Specifies mandatory functions and their accuracy

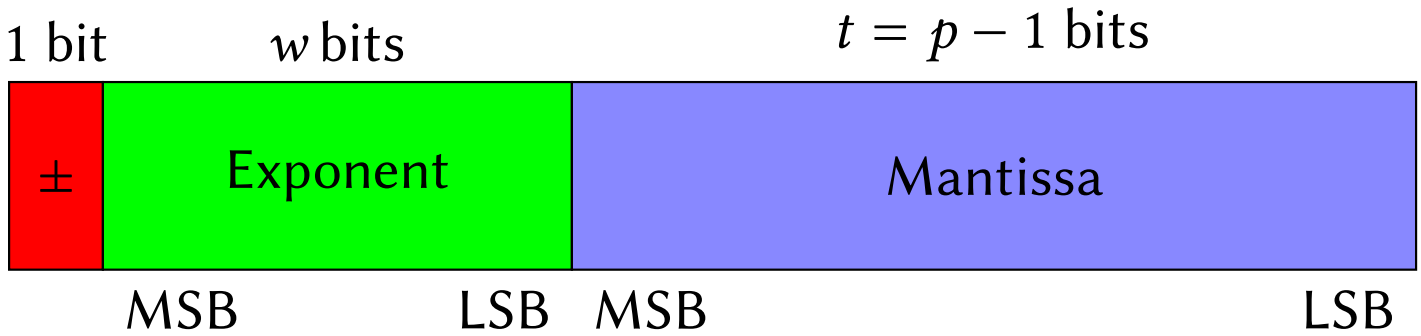
Machine, Arithmetic	β	t	$e_{\min}$	$e_{\max}$
IEEE half (not basic)	2	10	-14	15
IEEE single	2	23	-126	127
IEEE double	2	52	-1022	1023
IEEE extended	2	112	-16382	16383
bfloat16[15]	2	7	-126	127

- bfloat16 does not belong to the standard
  but available on Intel, ARM, Nvidia and Google Hardware
  applications are mostly ML and AI
- The standard also specifies a decimal format.

[15] Nicolas John Higham 2020; Wang and Kanwar 2019.

[16] IEEE Computer Society 2008; ISO/IEC10967-1 2012.

IEEE-754 Encoding[18]



Number	Encoding		
0.0	0	000 000 00	000 000 000 000 000 000 000 00
1.0	0	011 111 11	000 000 000 000 000 000 000 00
∞	0	111 111 11	000 000 000 000 000 000 000 00
NaN	0	111 111 11	000 000 000 000 000 000 000 01

Table: Sample encodings for 32 Bit ($w = 8, t = 23$)[17]

[17] Schmidt 2020.

[18] IEEE Computer Society 2008.

C/C++ Code for Analysis

```cpp
#include <iostream>
#include <bitset>
typedef union {
  float value;
  struct {
    unsigned int mantissa : 23;
    unsigned int exponent : 8;
    unsigned int sign     : 1;
  } raw;
} number;
int main() {
  number x;
  x.value = 1.0;
  std::bitset<1> s(x.raw.sign);
  std::bitset<8> e(x.raw.exponent);
  std::bitset<23> m(x.raw.mantissa);
  std::cout << s.to_string() << e.to_string() << m.to_string();
  // 0 01111111 00000000000000000000000
  return 0;
}
```

Listing: Bit representation of numbers

Arithmetic Operations and Rounding[19]

- Let $\mathrm{fl}(x)$ be the conversion of x to floating point
  It must hold
$$\mathrm{fl}(x) = x(1+\delta), \quad |\delta| \leqslant \frac{\varepsilon_M}{2}$$
- For arithmetic Operations we have
$$\mathrm{fl}(x \operatorname{op} y) = (x \operatorname{op} y)(1+\delta), \quad |\delta| \leqslant \frac{\varepsilon_M}{2}$$
  Für $\operatorname{op} \in \{+, -, \cdot, /\}$
- Hence relative errors are bounded (but not absolute errors)!

Operations are done in infinite precision and then rounded correctly!

[19] Nicholas John Higham 2002.

Use-Case: Variances[20]

$$\frac{1}{n-1}\sum_{k=1}^{n}\left(x_k - \frac{1}{n}\sum_{\ell=1}^{n} x_\ell\right)^2 \quad \text{vs.} \quad \frac{1}{n-1}\left(\sum_{k=1}^{n} x_k^2 - \frac{1}{n}\left(\sum_{\ell=1}^{n} x_\ell\right)^2\right)$$

```
1  function variances()
2    x::Float32 = 10000.0
3    y::Float32 = 10001.0
4    z::Float32 = 10002.0
5    mean::Float32 = (x+y+z)/3
6    s2::Float32 = ((x-mean)^2 + (y-mean)^2 + (z-mean)^2)/2
7    s1::Float32 = (x^2 + y^2 + z^2 - (x + y + z)^2/3)/2
8    println("Two pass version: ", s2)    # 1.0
9    println("Single pass version: ", s1) # 0.0
10 end
```

Listing: Computing Variances (Julia)

[20] Nicholas John Higham 2002; The Julia Project 2019; Chan, Golub and LeVeque 1983.

Use-Case: Angles[21]

$$\arccos\left(\frac{\langle x, y\rangle}{\|x\|\|y\|}\right) \quad \text{vs.} \quad \arctan\left(\|x \times y\|, \langle x, y\rangle\right)$$

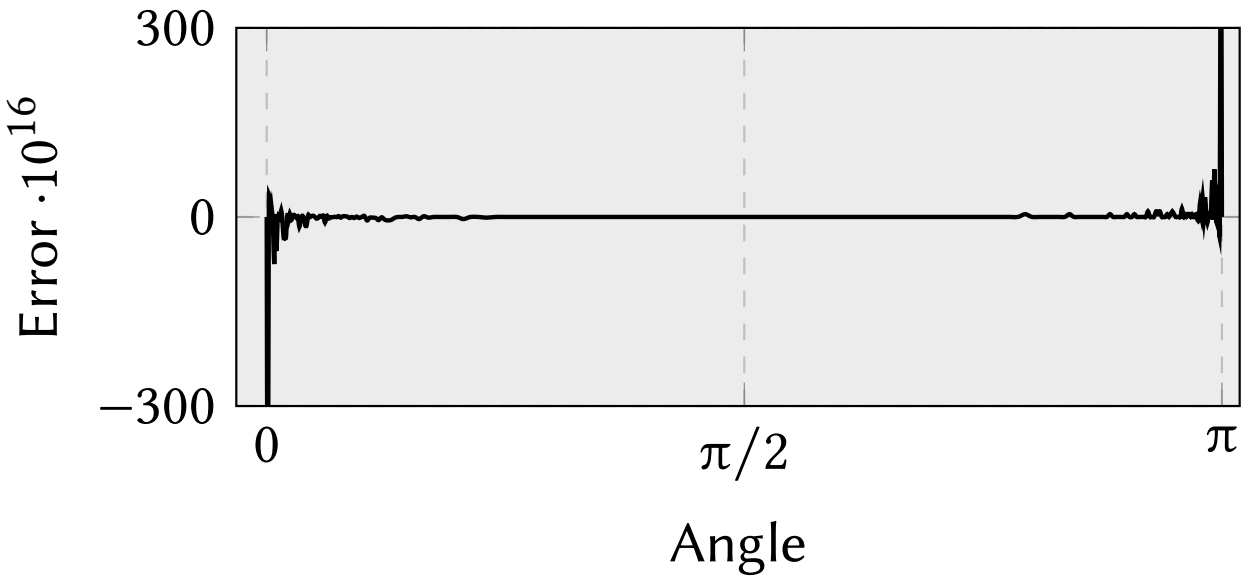


Figure: Error in arccos Formula

[21] Stackexchange 2017.

Mixed Precision Algorithms[22]

- Generic idea, works with lots of algorithms
- Approximate solution with low precision first
- Refine solution in higher precision in subsequent step

Algorithmus 2: Mixed Precision Solver for $Ax = b$

1 (ε_{SP}): $L \cdot U \leftarrow P \cdot A$
2 (ε_{SP}): solve $L \cdot y = P \cdot b$
3 (ε_{SP}): solve $U \cdot x = y$
4 **while** *not converged* **do**
5 (ε_{DP}): $r \leftarrow b - A \cdot x$
6 (ε_{SP}): solve $L \cdot y = P \cdot r$
7 (ε_{SP}): solve $U \cdot z = y$
8 (ε_{DP}): $x \leftarrow x + z$
9 **end while**

[22] Buttari et al. 2008; Baboulin et al. 2009; Abdelfattah et al. 2020.

Rules of Thumb

1. Avoid substraction of numbers that are inaccurate
   (c.f. catastrophic cancellation)
2. Avoid large numbers in magnitude compared to final result
3. Look for alternative formulations[23]
4. Formulate increments as new = old + small
5. Favour well-conditioned transforms
   (e.g. orthogonal matrices)
6. Avoid overflows and underflows

[23] Panchekha et al. 2020.

Thank You!

Literature I


Ahmad Abdelfattah et al. ‘A Survey of Numerical Methods Utilizing Mixed Precision Arithmetic’. In: (13th July 2020). arXiv: 2007.06674v1 [cs.MS].


Marc Baboulin et al. ‘Accelerating scientific computations with mixed precision algorithms’. In: *Computer Physics Communications* 180.12 (2009), pp. 2526–2533.


Thomas Bedürftig and Roman Murawski. *Philosophie der Mathematik*. De Gruyter, 2019.


Nelson H. F. Beebe. *The Mathematical-Function Computation Handbook*. Springer-Verlag GmbH, 8th Sept. 2017.


Alfredo Buttari et al. ‘Using mixed precision for sparse matrix computations to enhance the performance while achieving 64-bit accuracy’. In: *ACM Transactions on Mathematical Software (TOMS)* 34.4 (2008), pp. 1–22.

Literature II


Tony F. Chan, Gene H. Golub and Randall J. LeVeque. 'Algorithms for computing the sample variance: Analysis and recommendations'. In: *The American Statistician* 37.3 (1983), pp. 242–247.


Development of ternary computers at Moscow State University. 2020. (Visited on 05/07/2020).


Alan Edelman. *When is $x * (1/x) \neq 1$?*


Euclid. *The Thirteen Books of Euclid's Elements*. 3 vols. Dover Publications Inc, 2000.


David Goldberg. 'What Every Computer Scientist Should Know About Floating-Point Arithmetic'. In: *Computing Surveys* (1991).


Hidden Danger: The Problem. 2022. (Visited on 15/06/2022).


Nicholas John Higham. *Accuracy and Stability of Numerical Algorithms*. 2nd ed. Society for Industrial and Applied Mathematics, 2002.

Literature III


Nicolas John Higham. *What Is Bfloat16 Arithmetic?* 2nd June 2020. (Visited on 05/07/2020).


IEEE Computer Society. *IEEE Standard for Floating-PointArithmetic*. Tech. rep. 29th Aug. 2008.


ISO/IEC10967-1. *Information technology — Language independent arithmetic*. Part 1: Integer and floating point arithmetic. Tech. rep. 15th July 2012.


William Morton Kahan. *How Futile are Mindless Assessments of Roundoff in Floating-Point Computation?* 11th Jan. 2006. (Visited on 05/07/2020).


Daniel Lemire. *Multiplying by the inverse is not the same as the division*. 12th Mar. 2019. (Visited on 05/07/2020).


Cleve Moler. *Floating Point Denormals, Insignificant But Controversial*. 21st July 2014. (Visited on 05/07/2020).


Jean-Michel Muller. *Arithmétique des Ordinateurs*. 1989.

Literature IV

- Pavel Panchekha et al. *Herbgrind: Find and fix floating-point problems*. 2020. (Visited on 21/08/2020).
- Geórge Polya. *Schule des Denkens*. A. Francke Verlag Tübingen und Basel, 2010.
- PseudoRandom. *Implementing the Exponential Function*. 29th June 2020. (Visited on 05/07/2020).
- H. Schmidt. *IEEE-754 Floating Point Converter*. 2020. (Visited on 05/07/2020).
- Randy J. Shepherd. *IEEE 754 Rules & Properties*. 12th Sept. 2016. (Visited on 05/07/2020).
- Stackexchange. *Numerically stable way of computing angles between vectors*. 23rd Aug. 2017. (Visited on 05/07/2020).
- [SW] The GCC developers, *GCC, the GNU Compiler Collection* version 9.3.0, 2019. URL: `https://gcc.gnu.org`, (visited on 05/07/2020).

Literature V

- [SW] The Julia Project, *The Julia Programming Language* version 1.3.0, 2019. URL: https://julialang.org/, (visited on 05/07/2020).
- [SW] The Python Project, *The Python Programming Language* version 3.7.7, 2020. URL: https://www.python.org, (visited on 05/07/2020).
- [SW] The Ruby Project, *The Ruby Programming Language* version 2.3.7, 2018. URL: https://www.ruby-lang.org/, (visited on 05/07/2020).
- Shibo Wang and Pankaj Kanwar. *BFloat16: The secret to high performance on Cloud TPUs*. 23rd Aug. 2019. (Visited on 05/07/2020).
- Wikipedia. *Zermelo-Fraenkel-Mengenlehre*. 2020. (Visited on 07/07/2020).