Fließkommazahlen

Laurent Hoeltgen

11. September, 2020

Warum das Ganze?

„ Selbst mit 10^7 Iterationen, wird das Residuum nie kleiner als 10^{-16}.
(Student an der B-TU) “

„ [...] if you were as naive as me and thought computer numbers would work just like real numbers. (Bosch colleague) “

Ausblick

Was sind eigentlich Zahlen?

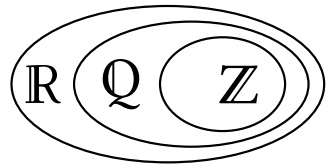
Unterschiede reelle Zahlen und Fließkommazahlen

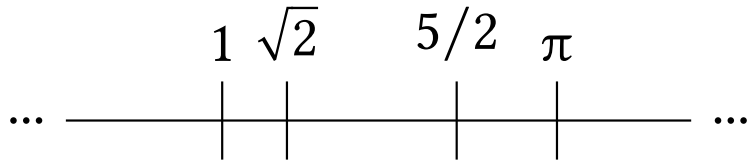
Fließkommazahlen Modell

Fließkommazahlen in der Praxis

Ontologie der Zahlen[1]

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





- $\mathbb{N}$: $\{\emptyset, \{\emptyset\}, \{\emptyset, \{\emptyset\}\}, ...\}$ (Zermelo-Fraenkel)
- $\mathbb{Z}$: Äquivalenzrelation auf $\mathbb{N} \times \mathbb{N}$
- $\mathbb{Q}$: Äquivalenzrelation auf $\mathbb{Z} \times \mathbb{Z} \setminus \{0\}$
- $\mathbb{R}$: Äquivalenzrelation von Paaren von Zahlenfolgen in $\mathbb{Q}$
- $\mathbb{C}$: Quotientenraum $\mathbb{R}[x]/(x^2 + 1)$

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

[1] Wikipedia 2020; Bedürftig und Murawski 2019.

1. Μονάς ἐστιν, καθ' ἣν ἕκαστον τῶν ὄντων ἓν λέγεται
   Einheit ist das, wonach jedes Ding eines genannt wird.
2. Ἀριθμὸς δὲ τὸ ἐκ μονάδων συγκείμενον πλῆθος.
   Zahl ist die aus Einheiten zusammengesetzte Vielheit.

(Euklid, Elemente VII, Defintion 1 & 2)

Zahlenkörper: Definition

Eine Menge G versehen mit einer Verknüpfung $\circ$ ist eine *abelsche Gruppe* wenn

1. G ist bezüglich $\circ$ abgeschlossen
2. $x \circ y = y \circ x$ und $(x \circ y) \circ z = x \circ (y \circ z), \quad \forall x, y, z \in \mathrm{G}$
3. Es gibt ein *neutrales* und ein *inverses* Element für jedes $x \in \mathrm{G}$

Eine Menge $\mathbb{K}$ versehen mit zwei Verknüpfungen $\oplus$ und $\odot$ ist ein *Körper* wenn

1. $(\mathbb{K}, \oplus)$ ist eine *abelsche Gruppe* mit neutralem Element $e_\oplus$
2. $(\mathbb{K} \setminus \{e_\oplus\}, \odot)$ ist eine *abelsche Gruppe* mit neutralem Element $e_\odot$
3. Es gelten die Distributivgesetze

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

Zahlen vs. Fließkommazahlen[2]

Eigenschaft	$\mathbb{Q}, \mathbb{R}, \mathbb{C}$	Fließkommazahlen
Abgeschlossen	✓	✓ / ✗
Kommutativ	✓	✓
Assoziativ	✓	✗
Einheitselement	✓	✓
Inverses	✓	✗
Distributiv	✓	✗
Monotonizität	✓	✓ / ✗

[2] Shepherd 2016; Goldberg 1991.

Verlust Mathematischer Eigenschaften I[3]

- *Abgeschlossen* unter + und ·, wenn man $\pm\infty$ und NaN erlaubt
- *Monotonizität* ist (fast) gegeben

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

  Ausnahme: $\pm\infty$ und NaN
- *Kommutativität* ist gewährleistet für + und ·

[3] Nicholas John Higham 2002.

Verlust Mathematischer Eigenschaften II[4]

- Reelle Zahlen sind *assoziativ*

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

- Fließkommazahlen sind nicht assoziativ

```
1 0.1 + (0.2 + 0.3) # 0.6
2 (0.1 + 0.2) + 0.3 # 0.6000000000000001
```

Listing: Keine Assoziativität von Fließkommazahlen (Python/Ruby)

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

Verlust Mathematischer Eigenschaften III[5]

- Für $+$ gibt es *immer* ein *Inverses*
- Für $\cdot$ gilt dies nicht

  10.0 ist Fließkommazahl, 0.1 ist *keine* Fließkommazahl

  $[0.1]_{10}$ in binär wäre $[0.\overline{00011}]_2$.
- x/y ist *nicht* das gleiche wie $x \cdot (1/y)$.

```
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: Probleme mit Inversen (Fortran)

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

Verlust Mathematischer Eigenschaften IV[6]

Reelle Zahlen sind *distributiv*

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

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

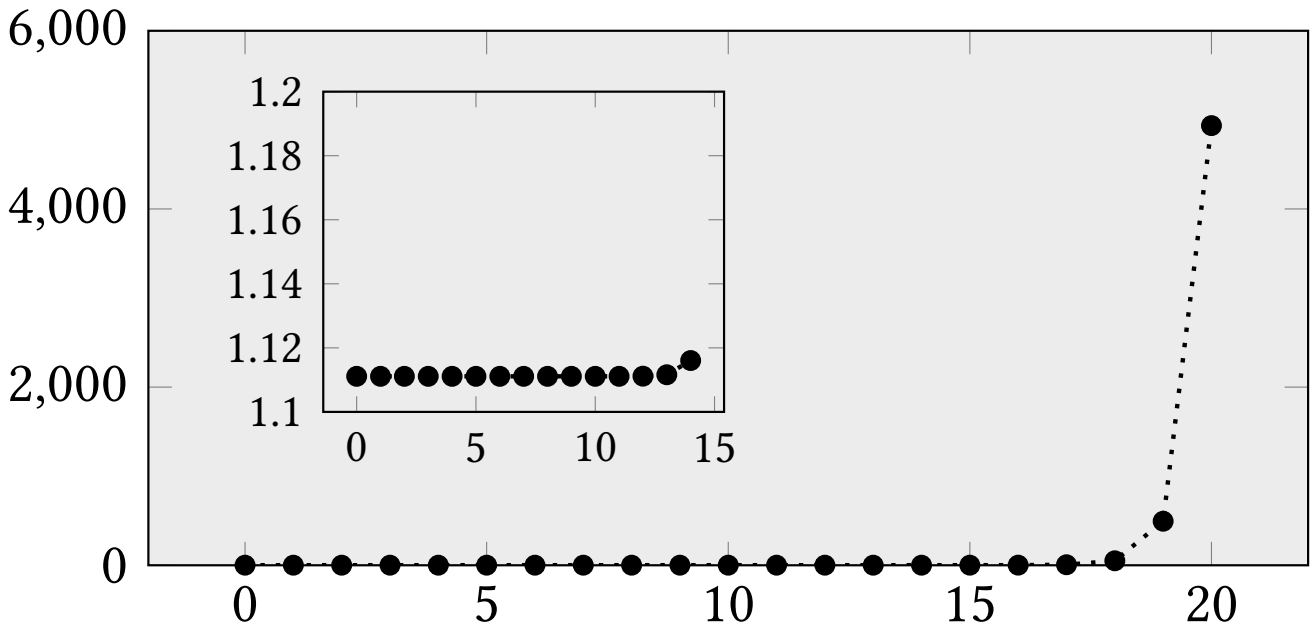
Listing: Keine Distributivität von Fließkommazahlen (Julia)

[6] Shepherd 2016; The Julia Project 2019.

Akkumulieren von Ungenauigkeit[7]

Wir betrachten folgende Iteration

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



Wir verlieren in jeder Iteration 1 Nachkommastelle an Genauigkeit.

[7] PseudoRandom 2020.

Trügerische Konvergenz[8]

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

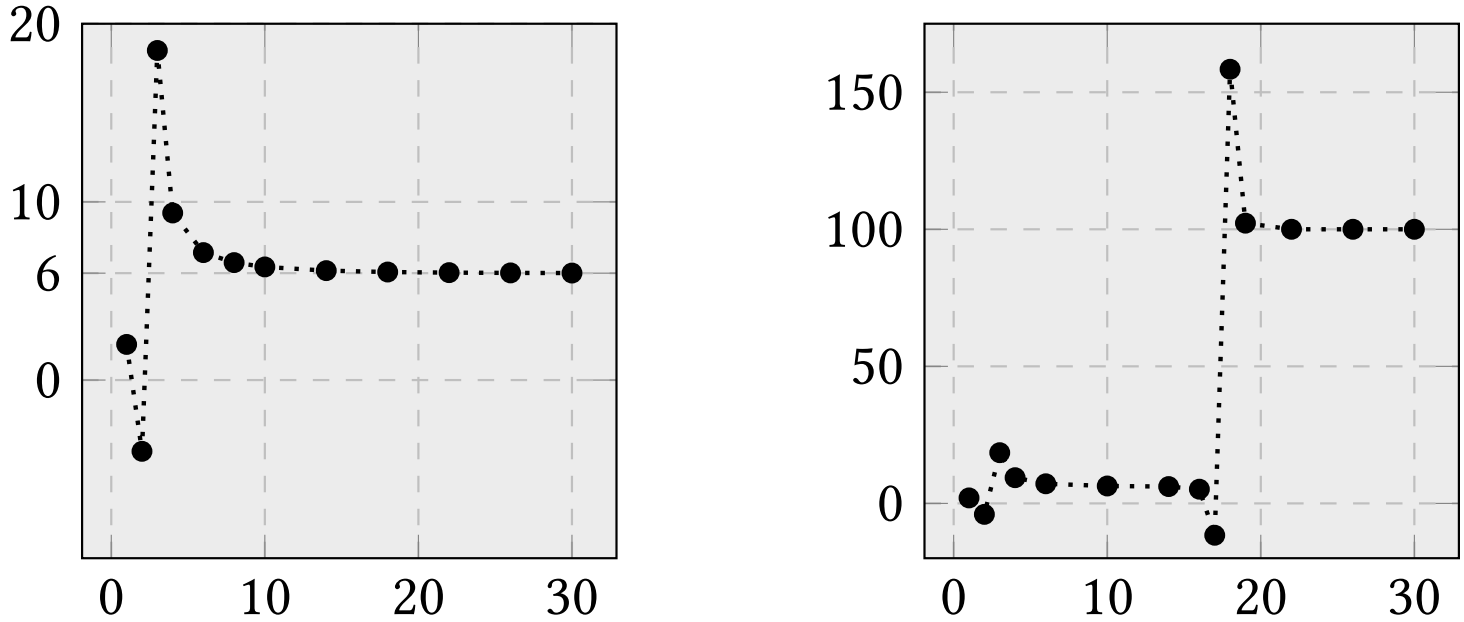


Abbildung: *Links:* Exakt (Konvergenz gegen 6), *Rechts:* Fließkommazahlen (Konvergenz gegen 100)

[8] Muller 1989.

Spaß mit Excel[9]

Eingabe	Resultat
$V = 4/3$	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

Tabelle: Inkonsistenzen im Zahlensystem von Excel

Funktioniert mit jeder Excel Version seit Excel 2000

[9] Kahan 2006.

Vergleich mit 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: Korrekte Ergebnisse

Fließkommazahlen Modell[10]

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

- $\beta \in \mathbb{N} \setminus \{0, 1\}$ nennt man *Basis*
- $t \in \mathbb{Z}$ nennt man *Präzision*
- $e \in [e_{\min}, e_{\max}]$ nennt man *Exponent*
- $m \in \{0, ..., \beta^t - 1\} \cap \mathbb{N}$, (bzw. $\{\beta^{t-1}, ..., \beta^t - 1\}$) nennt man *Mantisse*

m und e sind Eigenschaften der Zahl
β und t sind Eigenschaften des Zahlensystems

[10] Nicholas John Higham 2002.

Eigenschaften des Systems[11]

- $m \geqslant \beta^{t-1}$ gewährleistet Eindeutigkeit der Darstellung
- kleinste positive Zahl: $\beta^{e_{\min}-1}$
- größte positive Zahl: $\beta^{e_{\max}}(1-\beta^{-t})$
- *Maschinenepsilon* (Abstand 1 und nächster Zahl):

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

- *Unit in last place* (ulp): $\beta^e \cdot .00\ldots01 = \beta^{e-t}$.
- *Denormalisierte* Fließkommazahlen:

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

geringere Genauigkeit und stellen *Sicherheitsnetz* dar

[11] Nicholas John Higham 2002.

Fließkommazahlen in den letzten 50 Jahren[12]

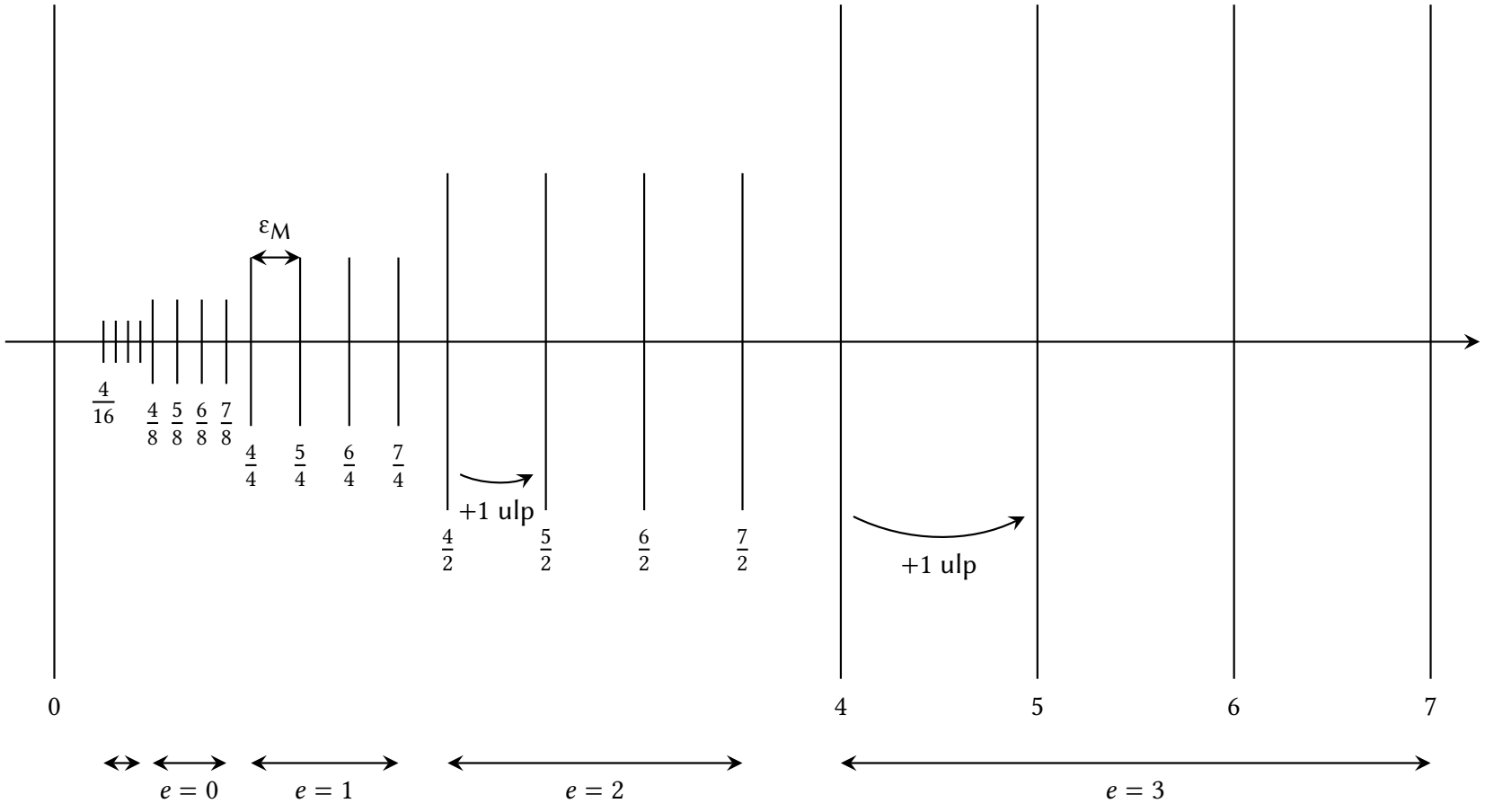
$$y = \pm m \cdot \beta^{e-t} \quad e \in [e_{\min}, e_{\max}]$$

Maschine, Arithmetik	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	-	-	-

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

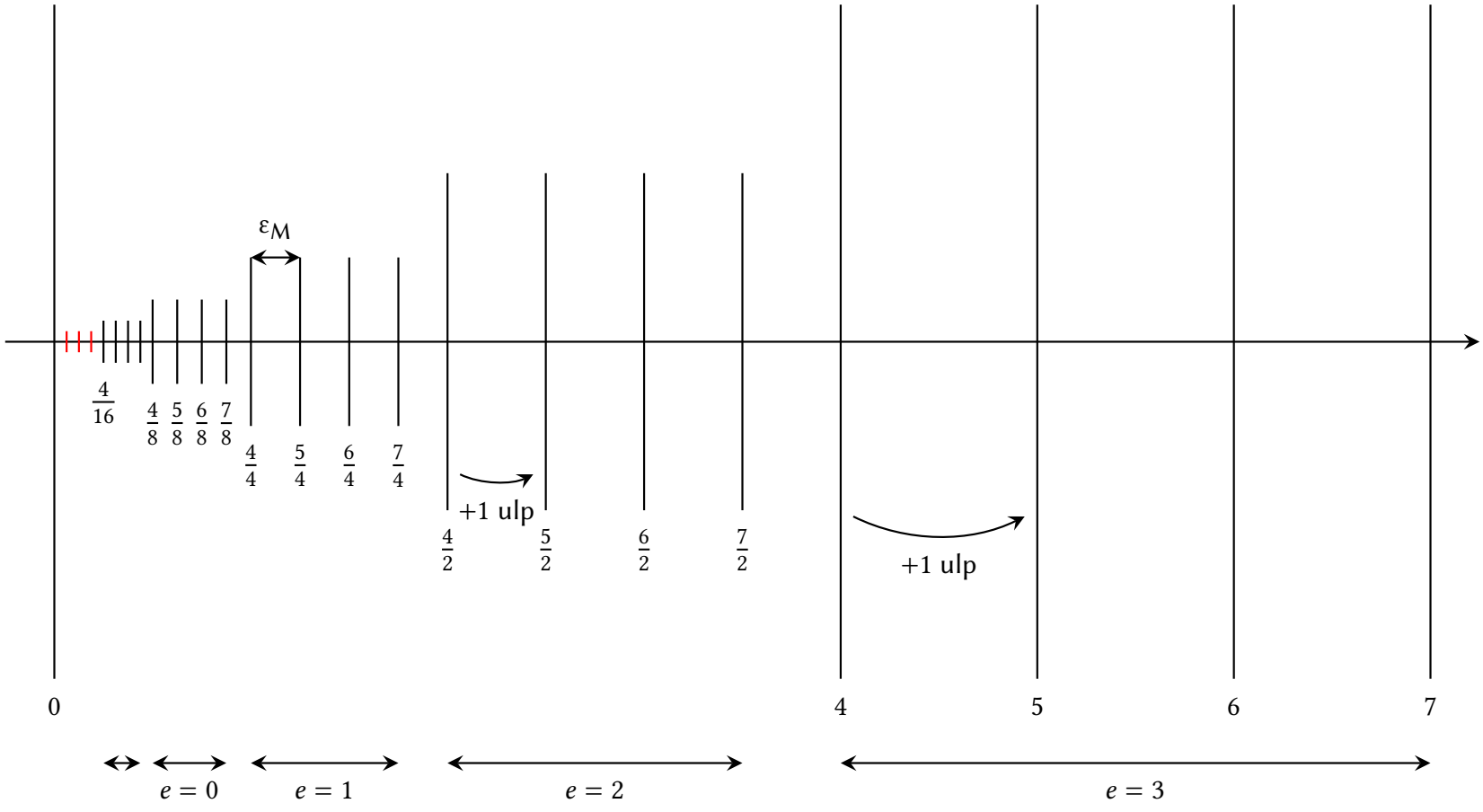
Beispiel

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



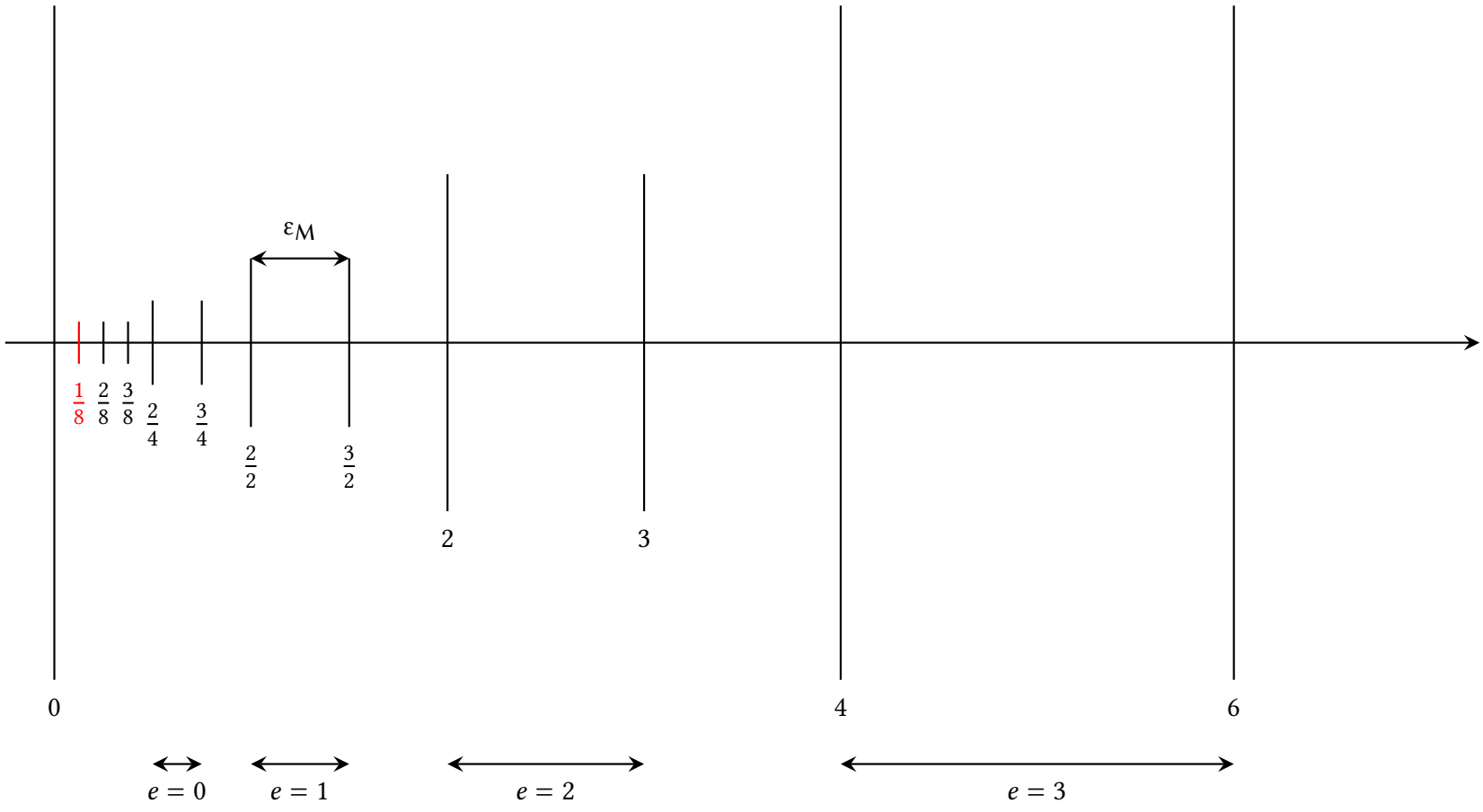
Beispiel

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



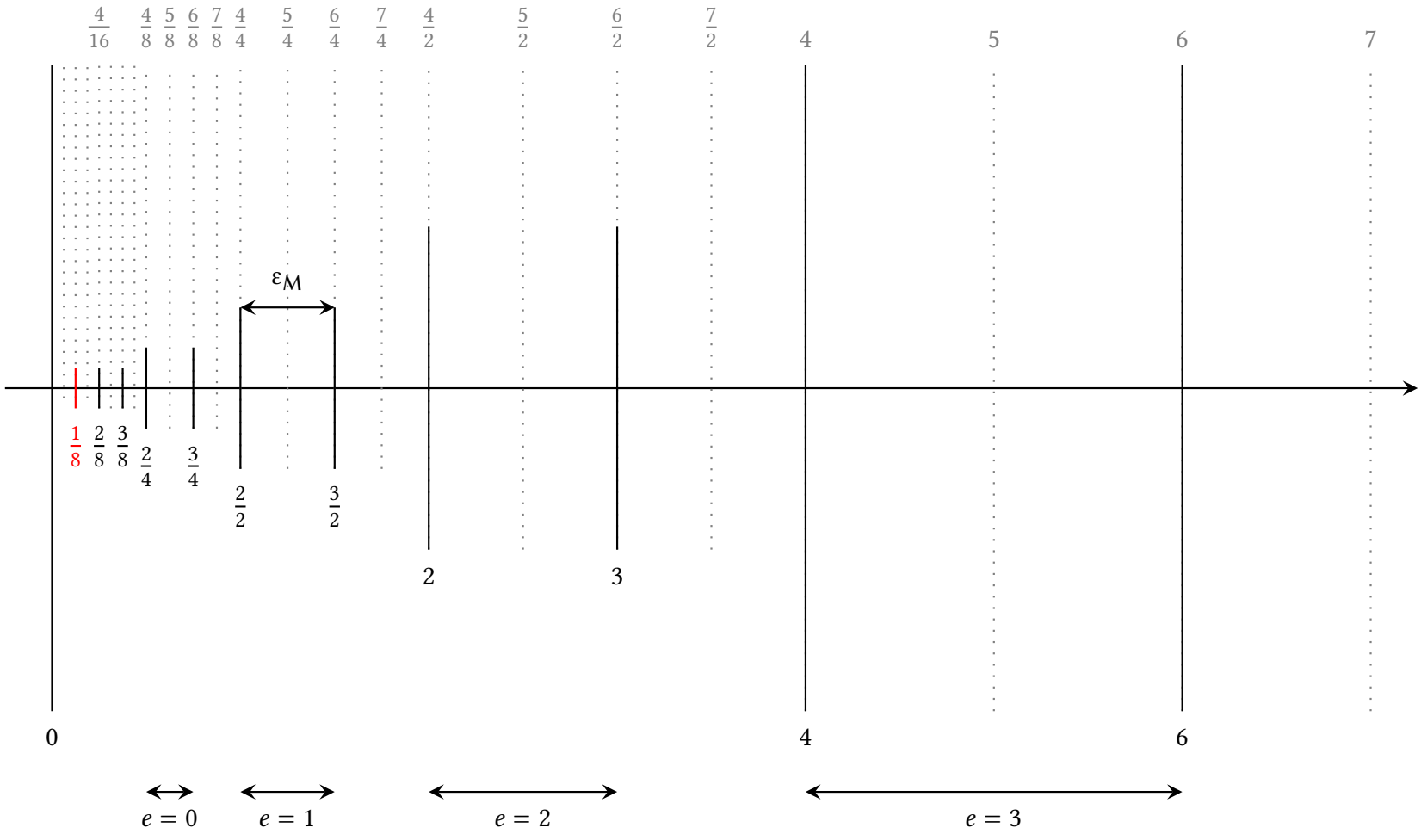
Beispiel

Wählt man stattdessen $t = 2$:



Beispiel

Wählt man stattdessen $t = 2$:



Sinn und Zweck von denormalisierten Zahlen[13]

Unser Fließkommazahlensystem:

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

Ohne denormalisierte Zahlen:

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

Algorithmus 1: Beispiel für Problematischen Code

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

[13] Moler 2014.

Der IEEE 754-2008 (ISO/IEC/IEEE 60559:2011) Standard[15]

- Der Standard gibt vor wie Fließkommazahlen abzuspeichern sind
- Die Rechenregeln sind in ISO/IEC 10967 spezifiziert

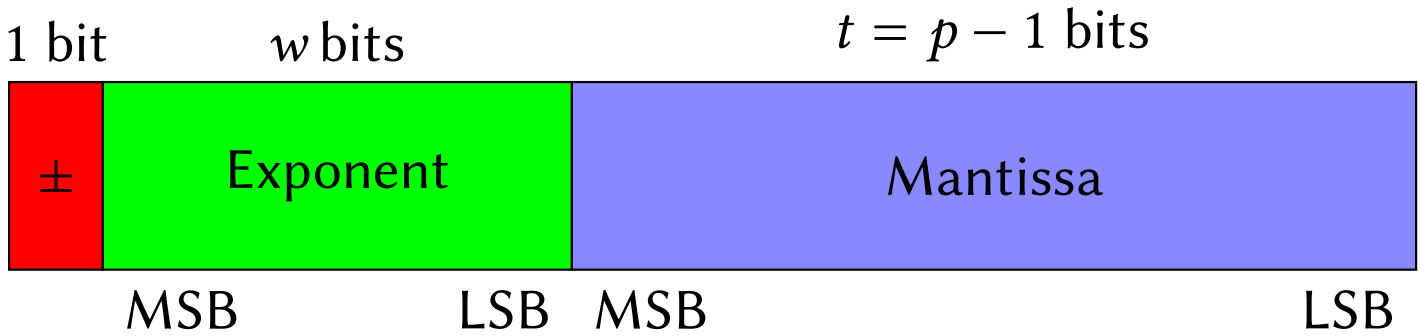
Maschine, Arithmetik	β	t	$e_{\min}$	$e_{\max}$
IEEE half	2	10	-14	15
IEEE single	2	23	-126	127
IEEE double	2	52	-1022	1023
IEEE extended	2	112	-16382	16383
bfloat16[14]	2	7	-126	127

- bfloat16 gehört (noch) nicht zum Standard
  verfügbar auf Intel, ARM, Nvidia und Google Hardware
  Anwendungsgebiete sind vor allem ML und AI

[14] Nicolas John Higham 2020; Wang und Kanwar 2019.

[15] IEEE Computer Society 2008; ISO/IEC 10967-1 2012.

IEEE-754 Kodierung[17]



Zahl	Kodierung		
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

Tabelle: Beispiel Kodierungen in 32 Bit ($w = 8, t = 23$)[16]

[16] Schmidt 2020.

[17] IEEE Computer Society 2008.

C/C++ Code zur Analyse

```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 Darstellung von Zahlen

Arithmetische Operationen und Runden[18]

- Sei $\mathrm{fl}(x)$ die Konvertierung von x in eine Fließkommazahl
  Es gilt
  $$\mathrm{fl}(x) = x(1+\delta), \quad |\delta| \leqslant \varepsilon_M$$
- Für arithmetische Operationen gilt

  $$\mathrm{fl}(x \,\mathrm{op}\, y) = (x \,\mathrm{op}\, y)(1+\delta), \quad |\delta| \leqslant \varepsilon_M$$

  Für $\mathrm{op} \in \{+, -, \cdot, /\}$

[18] Nicholas John Higham 2002.

Anwendung: Varianzen[19]

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

```
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: Varianzen berechnen (Julia)

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

Anwendung: Winkel[20]

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

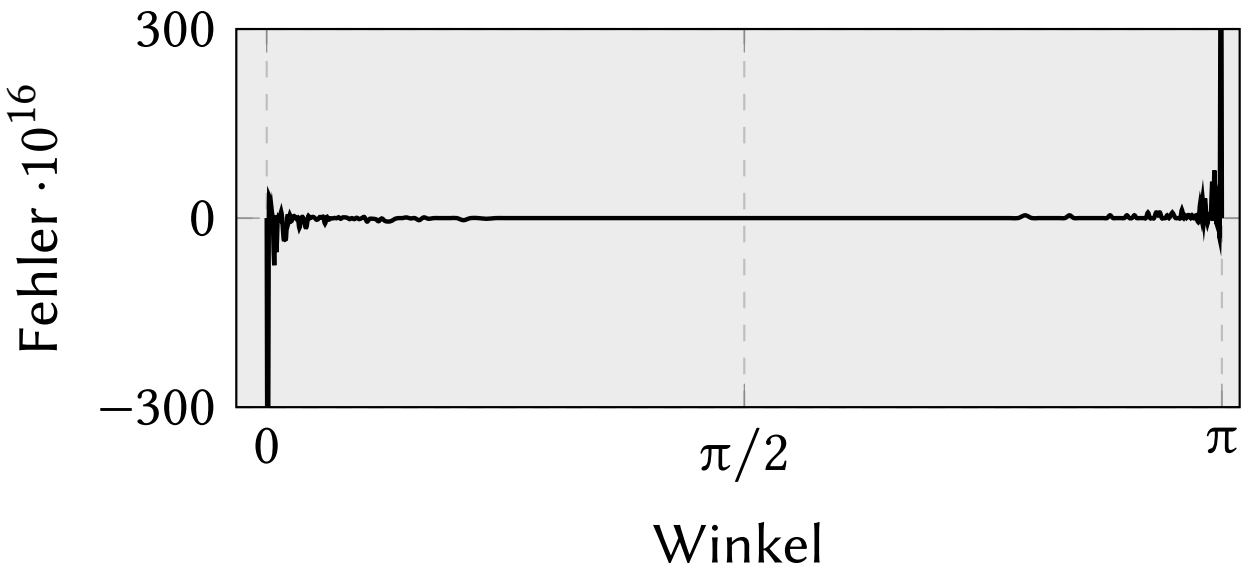


Abbildung: Fehler von arccos Formel bei Winkelberechnung

[20] Stackexchange 2017.

Mixed Precision Algorithmen[21]

- Generischer Ansatz, funktioniert mit vielen Algorithmen
- Löse Problem mit geringer Genauigkeit
- Verfeinere Lösung mit höherer Genauigkeit

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

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**

[21] Buttari u. a. 2008; Baboulin u. a. 2009; Abdelfattah u. a. 2020.

Allgemeine Tips

1. Vermeidet Subtraktion von Fehlerbehafteten Größen
2. Vermeidet große Werte relativ zum Endergebnis
3. Sucht nach alternativen Formulierungen[22]
4. Formuliert Inkrements als neu = alt + klein
5. Nutzt wohlkonditionierte Transformationen
   (z.B. orthogonale Matrizen)
6. Versucht overflows und underflows zu vermeiden

[22] Panchekha u. a. 2020.

Danke!

Literatur I


Ahmad Abdelfattah u. a. „A Survey of Numerical Methods Utilizing Mixed Precision Arithmetic“. In: (13. Juli 2020). arXiv: 2007.06674v1 [cs.MS].


Marc Baboulin u. a. „Accelerating scientific computations with mixed precision algorithms“. In: *Computer Physics Communications* 180.12 (2009), S. 2526–2533.


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


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


Alfredo Buttari u. a. „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), S. 1–22.

Literatur II


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


Development of ternary computers at Moscow State University. 2020. (Besucht am 05. 07. 2020).


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


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


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


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

Literatur III


Nicolas John Higham. *What Is Bfloat16 Arithmetic?* 2. Juni 2020. (Besucht am 05. 07. 2020).


IEEE Computer Society. *IEEE Standard for Floating-PointArithmetic.* Techn. Ber. 29. Aug. 2008.


ISO/IEC10967-1. *Information technology — Language independent arithmetic.* Part 1: Integer and floating point arithmetic. Techn. Ber. 15. Juli 2012.


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


Daniel Lemire. *Multiplying by the inverse is not the same as the division.* 12. März 2019. (Besucht am 05. 07. 2020).


Cleve Moler. *Floating Point Denormals, Insignificant But Controversial.* 21. Juli 2014. (Besucht am 05. 07. 2020).


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

Literatur IV

- Pavel Panchekha u. a. *Herbgrind: Find and fix floating-point problems*. 2020. (Besucht am 21. 08. 2020).
- Geórge Polya. *Schule des Denkens*. A. Francke Verlag Tübingen und Basel, 2010.
- PseudoRandom. *Implementing the Exponential Function*. 29. Juni 2020. (Besucht am 05. 07. 2020).
- H. Schmidt. *IEEE-754 Floating Point Converter*. 2020. (Besucht am 05. 07. 2020).
- Randy J. Shepherd. *IEEE 754 Rules & Properties*. 12. Sep. 2016. (Besucht am 05. 07. 2020).
- Stackexchange. *Numerically stable way of computing angles between vectors*. 23. Aug. 2017. (Besucht am 05. 07. 2020).
- Software The GCC developers, *GCC, the GNU Compiler Collection* Version 9.3.0, 2019. URL: `https://gcc.gnu.org`, (besucht am 05. 07. 2020).

Literatur V

- Software The Julia Project, *The Julia Programming Language* Version 1.3.0, 2019. URL: https://julialang.org/, (besucht am 05. 07. 2020).
- Software The Python Project, *The Python Programming Language* Version 3.7.7, 2020. URL: https://www.python.org, (besucht am 05. 07. 2020).
- Software The Ruby Project, *The Ruby Programming Language* Version 2.3.7, 2018. URL: https://www.ruby-lang.org/, (besucht am 05. 07. 2020).
- Shibo Wang und Pankaj Kanwar. *BFloat16: The secret to high performance on Cloud TPUs.* 23. Aug. 2019. (Besucht am 05. 07. 2020).
- Wikipedia. *Zermelo-Fraenkel-Mengenlehre.* 2020. (Besucht am 07. 07. 2020).