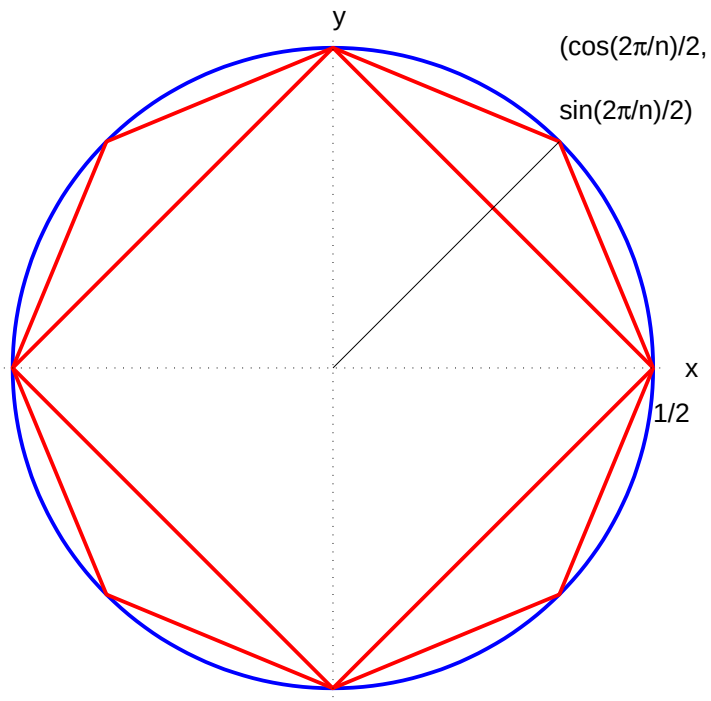
2 Gleitpunktarithmetik und Fehleranalyse

Einführendes Beispiel: Berechnung von π.

π = Umfang eines Kreises mit Radius $r = \frac{1}{2}$,

U_n = Umfang eines einbeschriebenen regelmäßigen n-Ecks $= n \sin(\pi/n)$.



Ein Algorithmus:

Klar: $\lim_{n\to\infty} U_n = \lim_{n\to\infty} n\sin(\pi/n) = \pi$ (unbrauchbar!)

Setze $A_n = U_{2^n}$ (Umfang des regelmäßigen 2^n-Ecks).
Dann gelten:

$$A_2 = U_4 = 4\sqrt{(1/2)^2 + (1/2)^2} = 2\sqrt{2},$$

$$A_{n+1} = 2^n\sqrt{2(1-\sqrt{1-(A_n/2^n)^2})}, \qquad n = 2, 3, \ldots \text{ (Rekursionsformel!)}$$

[ARCHIMEDES VON SYRAKUS (287–212 v. Chr.)]: $A_3 = 3.06\ldots,$
$A_4 = 3.12\ldots,$
$A_5 = 3.14\ldots.$

Eine Fehlerabschätzung

Zunächst gilt für $h > 0$:

$$|\sin(h) - h| \leq \frac{h^3}{6} \qquad \text{(Taylorformel).}$$

Setze $h = \pi/N$ (und multipliziere mit N):

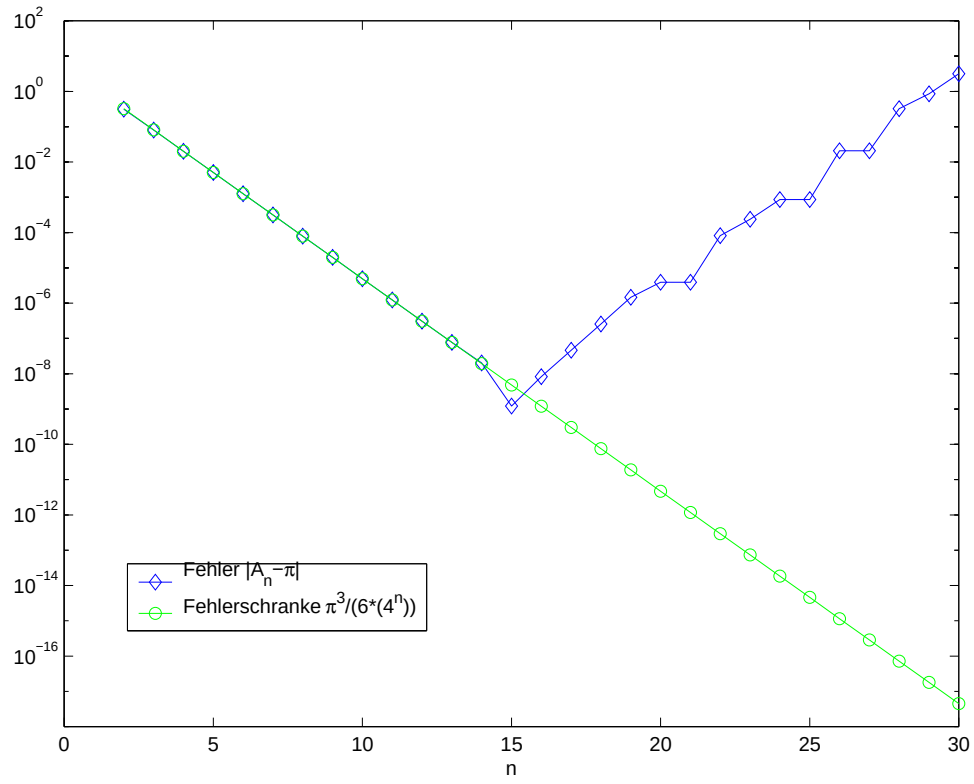
$$|N \sin(\pi/N) - \pi| \leq \frac{\pi^3}{6 \cdot N^2}.$$

D.h. $(N = 2^n)$:

$$|A_n - \pi| \leq \frac{\pi^3}{6 \cdot 4^n} \quad (< 10^{-10} \text{ für } n \geq 18).$$

Auf zum Rechner ...

Ernüchterung:



Die berechnete Folge $\{A_n\}$ verhält sich völlig anders als die „wirkliche“ Folge $\{A_n\}$! Wie ist das möglich?

Kapitel 2: Gleitpunktarithmetik und Fehleranalyse

2.1 Gleitpunktzahlen

Gleitpunktzahlen sind rationale Zahlen der Form

$$\pm(d_0.d_1d_2d_3\ldots d_{p-1})_b \cdot b^e, \qquad \text{wobei}$$

$b \in \mathbb{N}$ $(b > 1)$	Basis,
$m := (d_0.d_1d_2d_3\ldots d_{p-1})_b$	Mantisse (zur Basis b) und
$e \in \mathbb{Z}$, $e_{\min} \leq e \leq e_{\max}$	Exponent genannt werden.

Die Ziffern $d_0, d_1, d_2, \ldots, d_{p-1}$ sind jeweils ganze Zahlen zwischen 0 und $b-1$, womit für die Mantisse $0 \leq m \leq b(1-b^{-p})$ folgt.
Die Anzahl $p \in \mathbb{N}$ der Ziffern heißt Mantissenlänge.

Beispiel: Die Gleitpunktzahl $x = (0.10101)_2 \cdot 2^{-1}$ besitzt die Dezimaldarstellung

$$\left(0 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} + 0 \cdot 2^{-4} + 1 \cdot 2^{-5}\right) \cdot 2^{-1} = \frac{21}{64} = 0.328125.$$

Wahl der Basis: Verschiedene Werte von b sind möglich bzw. werden verwendet, etwa

- $b = 10$: die „Basis des täglichen Lebens“, wird auch intern von vielen Taschenrechnern verwendet;
- $b = 16$: in den 60er und 70er Jahren von IBM Mainframe-Computern (Baureihe 360/370) benutzt;
- $b = 3$: Forschungsrechner SETUN, Moskauer Staatsuniversität, Ende der 50er Jahre;
- $b = 2$: inzwischen auf allen Rechnern üblich, diese Wahl besitzt viele Vorteile sowohl technischer als auch mathematischer Natur.

Normalisierte Gleitpunktzahlen: Um möglichst viele Stellen einer Gleitpunktzahl in der Mantisse unterzubringen wird der Exponent so gewählt, dass die erste Ziffer der Mantisse (d.h. die erste *gültige* Ziffer) von Null verschieden ist. Solche Zahlen nennt man normalisiert; nicht normalisierte Zahlen werden auch subnormal oder denormalisiert genannt.
Im Fall $b = 2$ ist die erste (höchstwertige) Ziffer (=Bit) stets eine Eins, man kann sich deren explizite Darstellung daher sparen *(verstecktes Bit, hidden bit)*.

Maschinengenauigkeit: Die kleinste normalisierte Gleitpunktzahl mit Mantissenlänge p

$$x = \pm(d_0.d_1 d_2 \dots d_{p-1})_b \times b^e, \qquad d_0 \neq 0,$$

welche noch größer als Eins ist, lautet

$$(1.00\dots01)_b \times b^0 = 1 + b^{-(p-1)}.$$

Den Abstand $\varepsilon := b^{-(p-1)}$ dieser Zahl zu Eins bezeichnet man als Maschinengenauigkeit (machine epsilon).

Ulp: Allgemeiner definiert man für die obige Gleitpunktzahl

$$\mathrm{ulp}(x) := (0.00\ldots01)_b \times b^e = b^{-(p-1)} \times b^e = \varepsilon \cdot b^e.$$

Ulp steht für **u**nit in the **l**ast **p**lace, d.h. Stellenwert der letzten Ziffer, und gibt den Abstand zur betragsmäßig nächstgrößeren Gleitpunktzahl an.

Sonderfall Null ($b = 2$): Ist das höchstwertige Bit versteckt, so stellt eine Mantisse

$$(1.d_1 d_2 \ldots d_{p-1})_2, \quad d_1 = \cdots = d_{p-1} = 0$$

aus lauter Nullen nicht Null, sondern die Eins dar. Es ist daher erforderlich, einen Wert des Exponenten für die Darstellung der Null zu reservieren. (Ältere Implementierungen arbeiteten aus diesem Grund ohne verstecktes Bit, mussten dafür aber bei gleicher Wortbreite eine um Eins kürzere Mantissenlänge in Kauf nehmen.)

Auch die Frage, ob zwischen ± 0 unterschieden werden soll, hat praktische Konsequenzen (Kahan, 1987).

Ein Spielzeugbeispiel

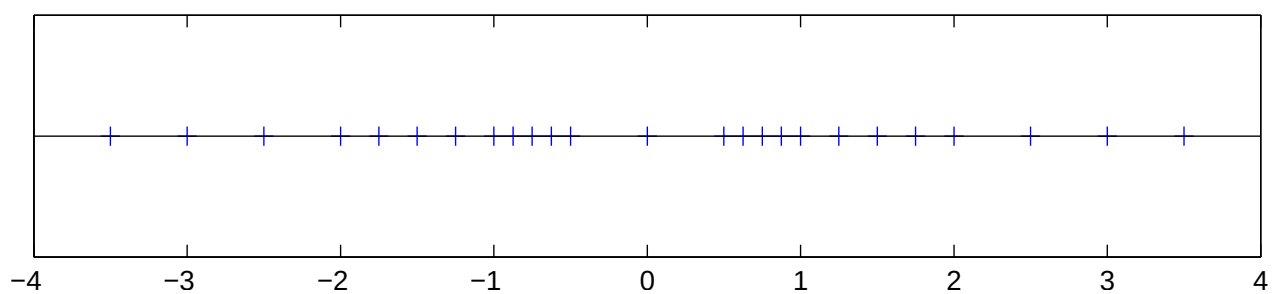
Wir betrachten das binäre Gleitpunktsystem bestehend aus Zahlen der Form

$$\pm(d_0.d_1d_2)_2 \times 2^e, \qquad e \in \{-1, 0, 1\}.$$

Die normalisierten Mantissen dieses Systems sind

$$(1.00)_2 = 1 \qquad (1.01)_2 = 1.25$$
$$(1.10)_2 = 1.5 \qquad (1.11)_2 = 1.75$$

Damit ergeben sich 24 normalisierte Gleitpunktzahlen, zusammen mit der Null also 25.



Charakteristische Größen dieses Systems:

Mantissenlänge: $p = 3$
größte normalisierte Zahl: $N_{\max} = (1.11)_2 \times 2^1 = 3.5$
kleinste normalisierte positive Zahl: $N_{\min} = (1.00)_2 \times 2^{-1} = 0.5$
Maschinengenauigkeit: $\varepsilon = (1.01)_2 - (1.00)_2 = 0.25$

$$\text{ulp}((d_0.d_1d_2)_2 \times 2^e) = \begin{cases} \varepsilon/2 & e = -1 \\ \varepsilon & e = 0 \\ 2\varepsilon & e = 1. \end{cases}$$

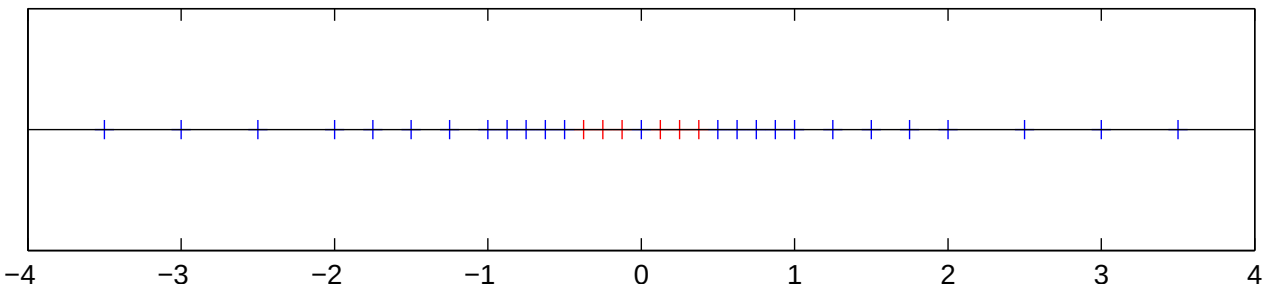
Wir bemerken ferner:

- Die Abstände zwischen den Gleitpunktzahlen nehmen von der Null weg zu.
- Aufgrund der Normalisierung klafft eine Lücke zwischen Null und der kleinsten normalisierten Zahl.

Subnormale Zahlen: Die eben erwähnte Lücke kann geschlossen werden, wenn wir für Zahlen mit Exponenten $e_{\min}$ auch denormalisierte Mantissen zulassen. In unserem Beispiel kommen dadurch die sechs Zahlen

$$\pm(0.01)_2 \times 2^{-1} = 0.125, \quad \pm(0.10)_2 \times 2^{-1} = 0.25, \quad \pm(0.11)_2 \times 2^{-1} = 0.375$$

hinzu. Der Abstand dieser Zahlen zur nächstgelegenen Gleitpunktzahl ist allerdings groß relativ zu deren Betrag.



2.2 Rundung

Sei $\mathbb{M}$:= Menge der Zahlen eines Gleitpunktsystems =: Maschinenzahlen. Liegt eine Eingangsgröße (etwa $1/10$ im Binärsystem) oder ein Zwischenergebnis x in $\mathbb{R} \setminus \mathbb{M}$, so muss hierfür ein Ersatz $\tilde{x} \in \mathbb{M}$ bestimmt werden, ein Vorgang den wir mit Rundung bezeichnen:

$$\mathrm{rd} : \mathbb{R} \to \mathbb{M}, \qquad x \mapsto \mathrm{rd}(x).$$

Üblich: Rundung zur nächstgelegenen Maschinenzahl: ist (hier $b = 10$)

$$x = \pm d_0.d_1 d_2 \ldots d_{p-1} d_p \ldots \times 10^e$$

mit $e_{\min} \leq e \leq e_{\max}$ aber möglicherweise unendlich langer Mantisse, so setzen wir

$$\mathrm{rd}(x) := \pm d_0.d_1 d_2 \ldots \tilde{d}_{p-1} \times 10^e, \quad \tilde{d}_{p-1} = \begin{cases} d_{p-1} & \text{falls } d_p \leq 4, \\ d_{p-1} + 1 & \text{falls } d_p \geq 5. \end{cases}$$

Ist $d_{p-1} = 9$, so entsteht ein Übertrag und d_{p-2}, möglicherweise auch $d_{p-3}, \ldots$ sowie e, müssen modifiziert werden.

Für $p = 4$ gilt etwa

$$\mathrm{rd}(4.4499) = 4.450 \times 10^0 \text{ und}$$
$$\mathrm{rd}(9.9999) = 1.000 \times 10^1.$$

Unschöne Eigenschaft dieser Rundung (hier stets $e = 0$):

$$\mathrm{rd}(1.0005) = 1.001,$$
$$\mathrm{rd}(\mathrm{rd}(1.000 + 0.0005) - 0.0005) = \mathrm{rd}(1.001 - 0.0005) = \mathrm{rd}(1.0005) = 1.001.$$

Dieses Phänomen bezeichnet man als Drift.

(Absoluter) Fehler bei Rundung: für eine Zahl $x = \pm m \times 10^e$ im normalisierten Bereich von $\mathbb{M}$ (d.h. $1 \leq m < 10, e_{\min} \leq e \leq e_{\max}$) gilt

$$|x - \mathrm{rd}(x)| \leq \tfrac{1}{2} \cdot 10^{-(p-1)} \times 10^e.$$

Allgemein: (Basis b, $1 \leq m < b$)

$$|x - \mathrm{rd}(x)| \leq \tfrac{1}{2} \cdot b^{-(p-1)} \times b^e = \tfrac{1}{2}\,\mathrm{ulp}(x).$$

Relativer Fehler bei Rundung:

$$\left|\frac{x - \mathrm{rd}(x)}{x}\right| \leq \left|\frac{\frac{1}{2} \cdot b^{-(p-1)} \times b^e}{m \times b^e}\right| \leq \tfrac{1}{2} \cdot b^{-(p-1)} = \tfrac{1}{2}\varepsilon =: u.$$

u heißt Rundungseinheit (unit roundoff). Anders formuliert

$$\mathrm{rd}(x) = (1 + \delta)x \quad \text{mit } |\delta| \leq u.$$

Vorsicht: manchmal wird auch u als Maschinengenauigkeit definiert.

2.3 Der IEEE-754 Standard

Nach eine langen Zeit des Wildwuchses im Bereich der Gleitpunkt-Arithmetik auf Computern fand Ende der 70er Jahre ein Standardisierungsprozess statt. Dieser führte schließlich 1985 zur Verabschiedung des *IEEE-754 Standards für binäre Gleitpunktarithmetik*[a], der inzwischen von nahezu allen Computerherstellern befolgt wird.

Der IEEE-Standard enthält drei wesentliche Forderungen:

Darstellung. Konsistente Darstellung von Gleitpunktzahlen auf allen konformen Maschinen

Rundung. Korrekt gerundete Gleitpunktoperationen bezüglich verschiedener Rundungsmodi

Ausnahmen. Wohldefiniertes Verhalten bei Ausnahmesituationen (wie etwa Division durch Null)

[a] IEEE = Institute for Electrical and Electronics Engineers

Sonderzahlen in IEEE 754:

$\pm\infty$: Manchmal ist es sinnvoll, mit Ausdrücken wie $1/0$ weiterzurechnen, anstatt das Programm abzubrechen. In IEEE-Arithmetik sind hierfür die Sonderzahlen $\pm\infty$ definiert, welche folgenden Konventionen unterliegen:

$$a + \infty = \infty \qquad (a > -\infty), \qquad a - \infty = -\infty \qquad (a < \infty),$$
$$a \cdot \infty = \infty \qquad (a > 0), \qquad a/0 = \infty \qquad (a > 0) \quad \text{usw.}$$

NaNs: Ist das Ergebnis einer arithmetischen Operation undefiniert, so wird dieses auf den Wert NaN (**N**ot **a** **N**umber) gesetzt. Beispiele: $\infty - \infty$, $0 \cdot \infty$, $0/0$ etc.

-0: IEEE-Arithmetik unterscheidet -0 von $+0$. So gilt $a/(-0) = -\infty$, $(a > 0)$ und umgekehrt wenn $a < 0$. **Achtung:** Es gilt zwar $0 = -0$, aber $\infty \neq -\infty$. Aus diesem Grund ist $a = b$ nicht äquivalent mit $1/a = 1/b$, etwa wenn $a = 0$ und $b = -0$.

Darstellung

IEEE-Arithmetik spezifiziert vier Formate für Gleitpunktzahlen:

- Single
- Double (optional, aber vom C-Standard verlangt), so gut wie überall verfügbar
- Single-extended (optional)
- Double-extended (optional)

Single-Format (FORTRAN: `REAL*4`, C: `float`) = 1 Wort = 32 Bits,

$\pm$	Exp.	Mantisse

0 1 9 31

Vorzeichen (1 Bit) 1 Bit, $0 = +$, $1 = -$

Exponent (8 Bits) Anstatt durch Vorzeichen-Betrag oder Zweierkomplement wird der Exponent verschoben dargestellt *(biased Exponent)*, d.h. der Wert e des Exponenten ergibt sich aus

$$e = E - 127, \quad 1 \leq E \leq 254, \text{ d.h. } -126 \leq e \leq 127,$$

wobei E die durch die 8 Bits dargestellte Zahl bezeichnet.
Die Werte $E = 0, 255$ sind reserviert für Sonderzahlen: $E = 0$ für subnormale Zahlen und Null, $E = 255$ für $\pm\infty$ und NaN.

Mantisse (23 Bits) Diese Ziffern stellen den Binärbruch dar.

Die IEEE Single-Zahlen im Überblick

Bitmuster E im Exponenten	dargestellte Gleitpunktzahl
$(00000000)_2 = 0$	$\pm(0.d_1 d_2 \ldots d_{23})_2 \times 2^{-126}$
$(00000001)_2 = 1$	$\pm(1.d_1 d_2 \ldots d_{23})_2 \times 2^{-126}$
$\vdots$	$\vdots$
$(01111111)_2 = 127$	$\pm(1.d_1 d_2 \ldots d_{23})_2 \times 2^{0}$
$(10000000)_2 = 128$	$\pm(1.d_1 d_2 \ldots d_{23})_2 \times 2^{1}$
$\vdots$	$\vdots$
$(11111110)_2 = 254$	$\pm(1.d_1 d_2 \ldots d_{23})_2 \times 2^{127}$
$(11111111)_2 = 255$	$\pm\infty$ falls $d_1 = \cdots = d_{23} = 0$, sonst NaN

Charakteristische Größen von IEEE Single:

Mantissenlänge: $p = 24$

größte normalisierte Zahl:

$$N_{\max} = (1.11\ldots1)_2 \times 2^{127} = 2(1-2^{-24}) \times 2^{127} \approx 2^{128} \approx 3.4 \times 10^{38}$$

kleinste normalisierte positive Zahl:

$$N_{\min} = (1.00\ldots0)_2 \times 2^{-126} = 2^{-126} \approx 1.2 \times 10^{-38}$$

kleinste positive Zahl:

$$M_{\min} = (0.0\ldots01)_2 \times 2^{-126} = 2^{-149} \approx 1.4 \times 10^{-45}$$

Maschinengenauigkeit:

$$\varepsilon = (1.0\ldots01)_2 - (1.0\ldots00)_2 = 2^{-23} \approx 1.2 \times 10^{-7}$$

Double-Format (FORTRAN: `REAL*8`, C: `double`) = 2 Worte = 64 Bits,

$\pm$	Exp.	Man-	tisse

0 1 11 32 63

d.h. 1 Bit Vorzeichen, 11-Bit Exponent und (1+)52-Bit Mantisse.

Charakteristika:

$$p = 53$$

$$e_{\min} = 1 - 1023 = -1022 \qquad e_{\max} = 2046 - 1023 = 1023$$

$$N_{\min} = 2^{-1022} \approx 2.2 \times 10^{-308} \qquad N_{\max} \approx 2^{1024} \approx 1.8 \times 10^{308}$$

$$M_{\min} = 2^{-1074} \approx 4.9 \times 10^{-324}$$

$$\varepsilon = 2^{-52} \approx 2.2 \times 10^{-16}$$

Rundung in IEEE Arithmetik

Zu $x \in \mathbb{R}$ seien $x_-, x_+ \in \mathbb{M}$ die nächstgelegenenen Maschinenzahlen kleiner bzw. größer als x. IEEE-Arithmetik definiert $\mathrm{rd}(x) := x$ falls $x \in \mathbb{M}$, andernfalls hängt der Wert $\mathrm{rd}(x)$ vom aktuell eingestellten Rundungsmodus ab, welcher einer der folgenden vier sein kann:

Abrunden. $\mathrm{rd}(x) = x_-$

Aufrunden. $\mathrm{rd}(x) = x_+$

Rundung zur Null. $\mathrm{rd}(x) = x_-$, falls $x \geq 0$ und $\mathrm{rd}(x) = x_+$ falls $x \leq 0$.

Rundung zur nächsten Maschinenzahl (Default). $\mathrm{rd}(x)$ erhält den näher an x liegenden Wert unter x_- und x_+. Liegt x *genau zwischen* x_- und x_+, so wird diejenige Zahl als $\mathrm{rd}(x)$ gewählt, *deren niedrigstwertiges Bit Null ist.* (Dies verhindert Drift.)

Weitere Ausnahme: $\mathrm{rd}(x) = \infty$ falls $x > N_{\max}$ und $\mathrm{rd}(x) = -\infty$ falls $x < -N_{\max}$.

Ausnahmesituationen (exceptions)

IEEE-Arithmetik definiert fünf Ausnahmesituationen sowie für jede dieser eine Standardreaktion:

invalid operation (ungültige Operation) $0/0$, ∞/∞, $\infty - \infty$, $\sqrt{-1}$ und dergleichen

division by zero (Division durch Null)

overflow (Exponentüberlauf) Ergebnis einer Operation größer als $N_{\max}$

underflow (Exponentunterlauf) Ergebnis einer Operation kleiner als $N_{\min}$ Das Weiterrechnen mit denormalisierten Maschinenzahlen bezeichnet man als gradual underflow.

inexact (ungenaues Ergebnis) Resultat keine Maschinenzahl (dies ist eigentlich keine Ausnahme)

IEEE-Philosophie bei Ausnahmesituationen

IEEE 754 fordert, dass beim Eintreten einer Ausnahmesituation ein Statusbit gesetzt wird, welches explizit wieder gelöscht werden muss (sticky bit). Ferner legt der Standard nahe, dass dem Programmierer die Möglichkeit gegeben wird, entweder die Behandlung dieser Ausnahmesituation durch speziellen Code selbst zu bestimmen (exception handling) oder die Ausnahmesituation zu ignorieren und weiterzurechnen (exception masking).

Dies gestattet es, nur in (seltenen) problematischen Fällen auf aufwendigere Varianten eines Programmcodes zurückzugreifen, um korrekte Behandlung des Rundefehlers zu gewährleisten.

Standardreaktionen	
invalid operation	Setze Ergebnis auf NaN
division by zero	Setze Ergebnis auf $\pm\infty$
overflow	Setze Ergebnis auf $\pm\infty$ oder $\pm N_{\max}$
underflow	Setze Ergebnis auf ± 0, $\pm N_{\min}$ oder subnormal
inexact	Setze Ergebnis auf korrekt gerundeten Wert

Exponentüberlauf kann durch geeignete Skalierung oft – auf Kosten eines harmlosen Unterlaufs – vermieden werden.

Beispiel: $c = \sqrt{a^2 + b^2}$ mit $a = 10^{60}$ und $b = 1$ (Rechnung mit vier Dezimalstellen in Mantisse und zwei Dezimalstellen im Exponent).

Standardauswertung verursacht Überlauf. Besser:

$$c = s\sqrt{(a/s)^2 + (b/s)^2} \quad \text{mit } s = \max\{|a|, |b|\}.$$

2.4 Korrekt gerundete Gleitpunktarithmetik

Die Machinenzahlen $\mathbb{M}$ sind bezüglich der elementaren arithmetischen Operationen (Addition, Subtraktion, Multiplikation und Division) nicht abgeschlossen (selbst wenn wir für die Exponenten beliebige Werte erlauben).

Beispiele:

- $x = 1.1 \cdot 10^0$ ist eine Gleitpunktzahl zur Basis 10 mit der Mantissenlänge 2, während $x \cdot x = 1.21 \cdot 10^{-1}$ eine dreistellige Mantisse besitzt.
- Im IEEE-Single Format sind 1 und 2^{-24} beides Maschinenzahlen, deren Summe $1 + 2^{-24}$ hingegen nicht.

Für jede der Operationen $\circ \in \{+, -, \cdot, /\}$ wird die entsprechende korrekt gerundete Gleitpunktoperation definiert durch

$$\mathrm{fl}(x \circ y) := \mathrm{rd}(x \circ y), \qquad x, y \in \mathbb{M}.$$

Für alle $x, y \in \mathbb{M}$ gilt daher, falls weder Unter- noch Überlauf eintritt,

$$\mathrm{fl}(x \circ y) = (1 + \delta)(x \circ y) \text{ mit } |\delta| \leq u.$$

Auf dieser Annahme fußt der Großteil moderner Rundungsfehleranalyse.

Man beachte aber, dass die neuen Operationen den klassischen Gesetzen der Arithmetik (wie etwa den Kommutativ-, Assoziativ- und Distributivgesetzen) nicht mehr genügen.
Z.B. in vierstelliger Gleitpunktarithmetik zur Basis 10:
$x = 1.234 \cdot 10^3$, $y = 1.234 \cdot 10^{-1} \in \mathbb{M}$, $x + y = 1.2341234 \cdot 10^3$, d.h. $\mathrm{fl}(x + y) = 1.234 \cdot 10^3$, was $\mathrm{fl}(x + y) = x$ bedeutet, obwohl $\mathrm{rd}(y) \neq 0$.

Gerundete Arithmetik im IEEE-Standard

IEEE 754 verlangt folgende korrekt gerundete Operationen

- die vier Grundrechenarten
- Quadratwurzel und Rest bei Division
- Formatkonvertierungen

Die korrekte Rundung richtet sich nach dem Zielformat, was je nach Variablentyp oder aktueller Hardware (Akkumulator, Register oder Speicherzelle) unterschiedlich sein wird.

Warum ist korrekte Rundung so wichtig?

Man betrachte etwa die folgenden vier Fragen:

Frage 1: Gilt $\mathrm{fl}(1 \cdot x) = x$ für $x \in \mathbb{M}$?

Frage 2: Gilt $\mathrm{fl}(x/x) = 1$ für $x \in \mathbb{M}$, $x \neq 0$, x endlich ?

Frage 3: Gilt $\mathrm{fl}(0.5 \cdot x) = \mathrm{fl}(x/2)$ für $x \in M$?

Frage 4: Folgt aus $\mathrm{fl}(x - y) = 0$ für $x, y \in \mathbb{M}$ auch $x = y$?

In IEEE-Arithmetik kann man jede dieser Fragen bejahen. In den 60er und 70er Jahren existierte zu jede Frage ein (jeweils weit verbreitetes) Computersystem, bei welchem für bestimmte Daten die Antwort „nein“ lautete.

Insbesondere kann man für IEEE-Arithmetik Frage 4 bejahen aufgrund der Verwendung subnormaler Zahlen.

Ein Blick in die Implementierung: Addition und Subtraktion

Gegeben: zwei IEEE-Single Zahlen $x = m_x \times 2^{e_x}$, $y = m_y \times 2^{e_y}$. Gilt $e_x = e_y$, so ergibt sich $\mathrm{fl}(x+y)$ aus $(m_x + m_y) \times 2^{e_x}$ mit anschließender Normalisierung. Beispiel: $3+2$:

$$\begin{aligned}
& (1.10000000000000000000000)_2 \times 2^1 \\
+ \quad & (1.00000000000000000000000)_2 \times 2^1 \\
= \quad & (10.10000000000000000000000)_2 \times 2^1 \\
\text{Normalisierung:} \quad & (1.01000000000000000000000)_2 \times 2^2.
\end{aligned}$$

Ist $e_x > e_y$, so müssen die Mantissen zuerst angepasst werden, z.B. bei $3 + 3/4$:

$$\begin{aligned}
& (1.10000000000000000000000)_2 \times 2^1 \\
+ \quad & (0.01100000000000000000000)_2 \times 2^1 \\
= \quad & (1.11100000000000000000000)_2 \times 2^1.
\end{aligned}$$

Hilfsziffern (guard digits)

Betrachte die Operation $3 + 3 \times 2^{-23}$:

$$
\begin{array}{rll}
 & (\ 1.10000000000000000000000 &)_2 \times 2^1 \\
 & +(\ 0.00000000000000000000001|1 &)_2 \times 2^1 \\
 & =(\ 1.10000000000000000000001|1 &)_2 \times 2^1 \\
\text{Abgerundet:} & (\ 1.10000000000000000000001 &)_2 \times 2^1 \\
\text{Aufgerundet:} & (\ 1.10000000000000000000010 &)_2 \times 2^1.
\end{array}
$$

In diesem Fall muss gerundet werden, da das Ergebnis keine Maschinenzahl ist. Allerdings erfordert die Berechnung der korrekt gerundeten Resultats eine Hilfsziffer rechts vom niedrigstwertigen Bit.

Bei der Rundung zur nächstgelegenen Maschinenzahl würde hier aufgerundet (warum?).

Auslöschung

Wir betrachten die Subtraktion der (benachbarten) Zahlen $x = 1$ und $y = (1.11\ldots1)_2 \times 2^{-1}$.

$$
\begin{aligned}
& (\ 1.00000000000000000000000 \quad)_2 \times 2^0 \\
-&(\ 0.11111111111111111111111|1 \quad)_2 \times 2^0 \\
=&(\ 0.00000000000000000000000|1 \quad)_2 \times 2^0 \\
\text{Normalisierung:} \quad & (\ 1.00000000000000000000000 \quad)_2 \times 2^{-24}
\end{aligned}
$$

Man spricht hier von Auslöschung, da sich alle Ziffern bis auf die letzte „wegheben“.

Auch hier ist eine Hilfsziffer unabdingbar für korrekte Rundung.

Notwendigkeit mehrerer Hilfsziffern

Betrachte $x - y$ mit $x = 1$ und $y = (1.00\ldots01)_2 \times 2^{-25}$. Bei der Verwendung von 25 Hilfsziffern erhalten wir

$$
\begin{aligned}
&(\ 1.00000000000000000000000| &&)_2 \times 2^0 \\
-&(\ 0.00000000000000000000000|0100000000000000000000001 &&)_2 \times 2^0 \\
=&(\ 0.11111111111111111111111|1011111111111111111111111 &&)_2 \times 2^0 \\
=&(\ 1.11111111111111111111111|0111111111111111111111110 &&)_2 \times 2^{-1} \\
=&(\ 1.11111111111111111111111 &&)_2 \times 2^{-1}
\end{aligned}
$$

(Der Rundungsmodus ist Rundung zur nächsten Maschinenzahl).

Weniger als 25 Hilfsziffern hätten hier nicht genügt, um das korrekt gerundete Ergebnis zu berechnen (nachprüfen!).

Bei Rechnern der Firma CRAY Research war bis vor kurzem die Subtraktion aufgrund fehlender Hilfsziffern nicht korrekt gerundet.

Man kommt aber mit weniger Hilfsbits aus:

Wir verwenden nun zwei Hilfsziffern und ein zusätzliches Hilfsbit, welches dann gesetzt wird, wenn beim Shiften der Mantisse mindestens ein von Null verschiedenes Bit verlorengegangen (d.h. jenseits der zweiten Hilfsziffer gewandert) ist. Dieses Bit setzen wir vor der Subtraktion an die dritte Hilfsziffer:

$$
\begin{array}{rll}
 & (\ 1.00000000000000000000000 &)_2 \times 2^0 \\
 & -(\ 0.00000000000000000000000|011 &)_2 \times 2^0 \\
 & =(\ 0.11111111111111111111111|101 &)_2 \times 2^0 \\
\text{Normalisierung:} & (\ 1.11111111111111111111111|01 &)_2 \times 2^{-1} \\
\text{Rundung:} & (\ 1.11111111111111111111111 &)_2 \times 2^{-1}
\end{array}
$$

Man kann zeigen, dass für korrekt gerundete Subtraktion nicht mehr als diese zwei Hilfsziffern und das Hilfsbit (sticky bit) benötigt werden.

Multiplikation und Division:

Hier ist ein Anpassen der Mantissen nicht notwendig: Multiplikation von $x = m_x \times 2^{e_x}$ mit $y = m_y \times 2^{e_y}$ ergibt

$$xy = (m_x m_y) \times 2^{e_x + e_y}.$$

Somit besteht die Multiplikationsoperation aus den drei Schritten Multiplikation der Operandenmantissen, Addition der Operandenexponenten und Normalisierung des Ergebnisses. (Analog bei Division).

Relative Geschwindigkeit von Multiplikation/Division im Vergleich zu Addition/Subtraktion: Im Prinzip gleich schnell in Hardware realisierbar, allerdings mit wesentlich mehr Aufwand.

Aktueller Kompromiss beim Chipentwurf: Multiplikation ungefähr so schnell wie Addition/Subtraktion, Division deutlich langsamer.

2.5 Numerische Stabilität und Fehleranalyse

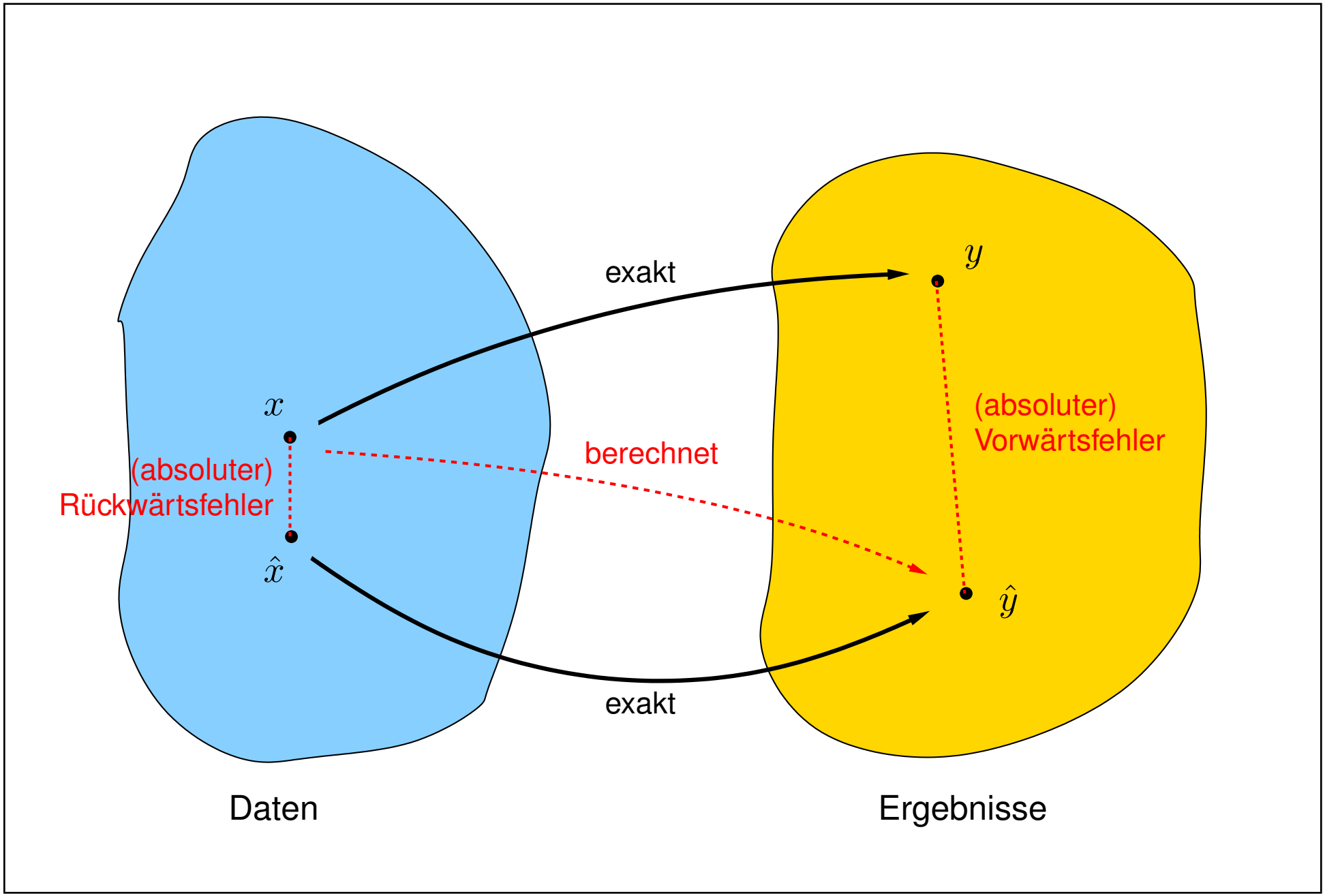
Es sei $\hat{y} = \mathrm{fl}(f(x))$ das in Gleitpunktarithmetik berechnete Ergebnis der Auswertung einer Funktion $y = f(x)$.

Wie beurteilt man die Qualität von $\hat{y}$?

- (Relativer) Vorwärtsfehler: $|(y - \hat{y})/y|$.
- (Relativer) Rückwärtsfehler: $|(x - \hat{x})/x|$, dabei ist $\hat{x}$ das (ein) Eingabedatum, das bei rundungsfreier Rechnung zu $\hat{y}$ führt: $f(\hat{x}) = \hat{y}$ (Rundungsfehler werden als Datenfehler interpretiert).

Mit Störungstheorie kann man Vorwärtsfehler durch Rückwärtsfehler abschätzen. Faustregel:

$$\textbf{Vorwärtsfehler} \lessapprox \textbf{Konditionszahl} \times \textbf{Rückwärtsfehler}.$$


exakt
y
x
(absoluter)
Vorwärtsfehler
berechnet
(absoluter)
Rückwärtsfehler
$\hat{x}$
$\hat{y}$
exakt
Daten
Ergebnisse

Ein Algorithmus heißt

vorwärtsstabil, wenn der Vorwärtsfehler „klein“ ist,

rückwärtsstabil, wenn der Rückwärtsfehler „klein“ ist; was „klein“ bedeutet, hängt vom Problem und der Maschinengenauigkeit ab.

Die Kondition(szahl) eines Problems (hat nichts mit Gleitpunktarithmetik zu tun!!) ist ein Maß dafür, wie empfindlich das Ergebnis auf Störungen der Daten reagiert.

Ein Problem ist gut (schlecht) konditioniert, wenn seine Konditionszahl klein (groß) ist.

Bestimme $y = f(x)$, Störung der Daten: Δx

$$\hat{y} = f(x + \Delta x) = f(x) + f'(x)\Delta x + \tfrac{1}{2} f''(\zeta)(\Delta x)^2.$$

Δx klein: $\hat{y} = f(x + \Delta x) \approx f(x) + f'(x)\Delta x = y + f'(x)\Delta x$ oder

$$\left| \frac{f(x + \Delta x) - f(x)}{f(x)} \right| = \left| \frac{\hat{y} - y}{y} \right| \approx \left| \frac{x f'(x)}{f(x)} \right| \left| \frac{\Delta x}{x} \right|.$$

(Relative) Konditionszahl von f an der Stelle x:

$$c_f(x) = \left| \frac{x f'(x)}{f(x)} \right|.$$

Beispiel. $f(x) = \log(x)$, d.h.: $c_f(x) = \left|\frac{x/x}{\log(x)}\right| = \left|\frac{1}{\log(x)}\right|$ moderat für sehr kleine und sehr große (positive) x, riesig für $x \approx 1$.

$x_1 = 0.01$: $\quad c_f(x_1) = 0.21715$,
$x_2 = 0.99$: $\quad c_f(x_2) = 99.4992$,
$x_3 = 100.$: $\quad c_f(x_3) = 0.21715$.

Wie wirkt sich eine relative Störung von $\varepsilon_x = (\Delta x)/x = 0.001$ aus?
Prognose:

$$\left|\frac{f(x_k + 0.001x_k) - f(x_k)}{f(x_k)}\right| \approx 0.001\, c_f(x_k) = 0.001 \left|\frac{1}{\log(x_k)}\right|.$$

k	rel. Fehler	Prognose
1	$2.1704 \cdot 10^{-4}$	$2.1715 \cdot 10^{-4}$
2	$9.9945 \cdot 10^{-2}$	$9.9499 \cdot 10^{-2}$
3	$2.1704 \cdot 10^{-4}$	$2.1715 \cdot 10^{-4}$

Allgemeiner: $y = f(x_1, x_2, \ldots, x_n)$. Absolute Störungen der Daten, Δx_k ($k = 1, 2, \ldots, n$), verursachen absoluten Fehler im Ergebnis:

$$\begin{aligned} \Delta y &= f(x_1 + \Delta x_1, \ldots, x_n + \Delta x_n) - f(x_1, \ldots, x_n) \sim \sum_{k=1}^{n} d_k \Delta x_k, \\ d_k &= \frac{\partial f(x_1, x_2, \ldots, x_n)}{\partial x_k} \quad \text{(absolute Konditionszahlen von } f\text{)}. \end{aligned}$$

Relative Störungen der Daten, $\varepsilon_k = \Delta x_k / x_k$ ($k = 1, 2, \ldots, n$), verursachen relativen Fehler im Ergebnis:

$$\begin{aligned} \varepsilon_y &= \frac{f(x_1 + \Delta x_1, \ldots, x_n + \Delta x_n) - f(x_1, \ldots, x_n)}{f(x_1, \ldots, x_n)} \sim \sum_{k=1}^{n} c_k \varepsilon_k, \\ c_k &= \frac{x_k}{f(x_1, x_2, \ldots, x_n)} \frac{\partial f(x_1, x_2, \ldots, x_n)}{\partial x_k} \end{aligned}$$

(relative Konditionszahlen von f).

Beispiele. (Grundoperationen)

- $y = f(x_1, x_2) = x_1 \cdot x_2$. D.h. $c_1 = 1$ und $c_2 = 1$ (unproblematisch).
- $y = f(x_1, x_2) = x_1/x_2$. D.h. $c_1 = 1$ und $c_2 = -1$ (unproblematisch).
- $y = f(x_1, x_2) = x_1 + x_2$. D.h. $c_1 = x_1/(x_1 + x_2)$ und $c_2 = x_2/(x_1 + x_2)$.
- $y = f(x_1, x_2) = x_1 - x_2$. D.h. $c_1 = x_1/(x_1 - x_2)$ und $c_2 = -x_2/(x_1 - x_2)$.

Bei den Operationen $\pm$ können die Konditionszahlen riesig werden:
$x_1 \approx -x_2$: Addition schlecht konditioniert.
$x_1 \approx x_2$: Subtraktion schlecht konditioniert. (Auslöschung!)

Etwa: $x_1 = 3.14159$, $x_2 = 3.14140$. $\Delta x_1 = 10^{-5}$, $\Delta x_2 = 2 \cdot 10^{-5}$, dh. $\varepsilon_1 \approx 3.18 \cdot 10^{-6}$, $\varepsilon_2 \approx 6.36 \cdot 10^{-6}$.
$y = x_1 - x_2 = 0.00019$ (Auslöschung führender Ziffern). $(x_1 + \Delta x_1) - (x_2 + \Delta x_2) = 0.00018$. Also $\varepsilon_y = 5.26 \cdot 10^{-2}$.
Prognose: $c_1 = 1.65 \cdot 10^4$, $c_2 = -1.65 \cdot 10^4$, $|\varepsilon_y| \approx |c_1\varepsilon_1 + c_2\varepsilon_2| \leq 1.65 \cdot 10^4 \left(3.18 \cdot 10^{-6} + 6.36 \cdot 10^{-6}\right) \approx 1.57 \cdot 10^{-1}$.

2.6 Ein Beispiel

Die quadratische Gleichung

$$x^2 - bx + c = 0$$

hat die Lösungen

$$x_{1/2} = \frac{b \pm \sqrt{b^2 - 4c}}{2}.$$

Für $b = 3.6678$ und $c = 2.0798 \cdot 10^{-2}$
erhält man nach Rechnung mit fünfstelliger Dezimalmantisse

$\tilde{x}_1 = 3.6673$ (rel. Fehler: $4.7 \cdot 10^{-6}$), $\tilde{x}_2 = 5.5 \cdot 10^{-4}$ (rel. Fehler: $3.0 \cdot 10^{-2}$).

Warum?

Schritt	Ergebnis	rel. Fehler
1. b^2	$1.3453 \cdot 10^{+1}$	$1.8 \cdot 10^{-5}$
2. $4c$	$8.3192 \cdot 10^{-3}$	0.0
3. $b^2 - 4c$	$1.3445 \cdot 10^{+1}$	$4.1 \cdot 10^{-5}$
4. $\sqrt{b^2 - 4c}$	$3.6667 \cdot 10^{+0}$	$9.3 \cdot 10^{-6}$
5. $b - \sqrt{b^2 - 4c}$	$1.1000 \cdot 10^{-3}$	$3.0 \cdot 10^{-2}$
6. $(b - \sqrt{b^2 - 4c})/2$	$5.5000 \cdot 10^{-4}$	$3.0 \cdot 10^{-2}$
5'. $b + \sqrt{b^2 - 4c}$	$7.3345 \cdot 10^{+0}$	$4.7 \cdot 10^{-6}$
6'. $(b + \sqrt{b^2 - 4c})/2$	$3.6673 \cdot 10^{+0}$	$4.7 \cdot 10^{-6}$
7. $x_2 = c/x_1$	$5.6713 \cdot 10^{-4}$	$1.1 \cdot 10^{-6}$

Beispiel aus der Einleitung dieses Kapitels:

$$A_{n+1} = 2^n \, [2 \underbrace{\left(1 - \sqrt{1 - (A_n/2^n)^2}\right)}_{\text{Auslöschung!}}]^{1/2}.$$

Setze

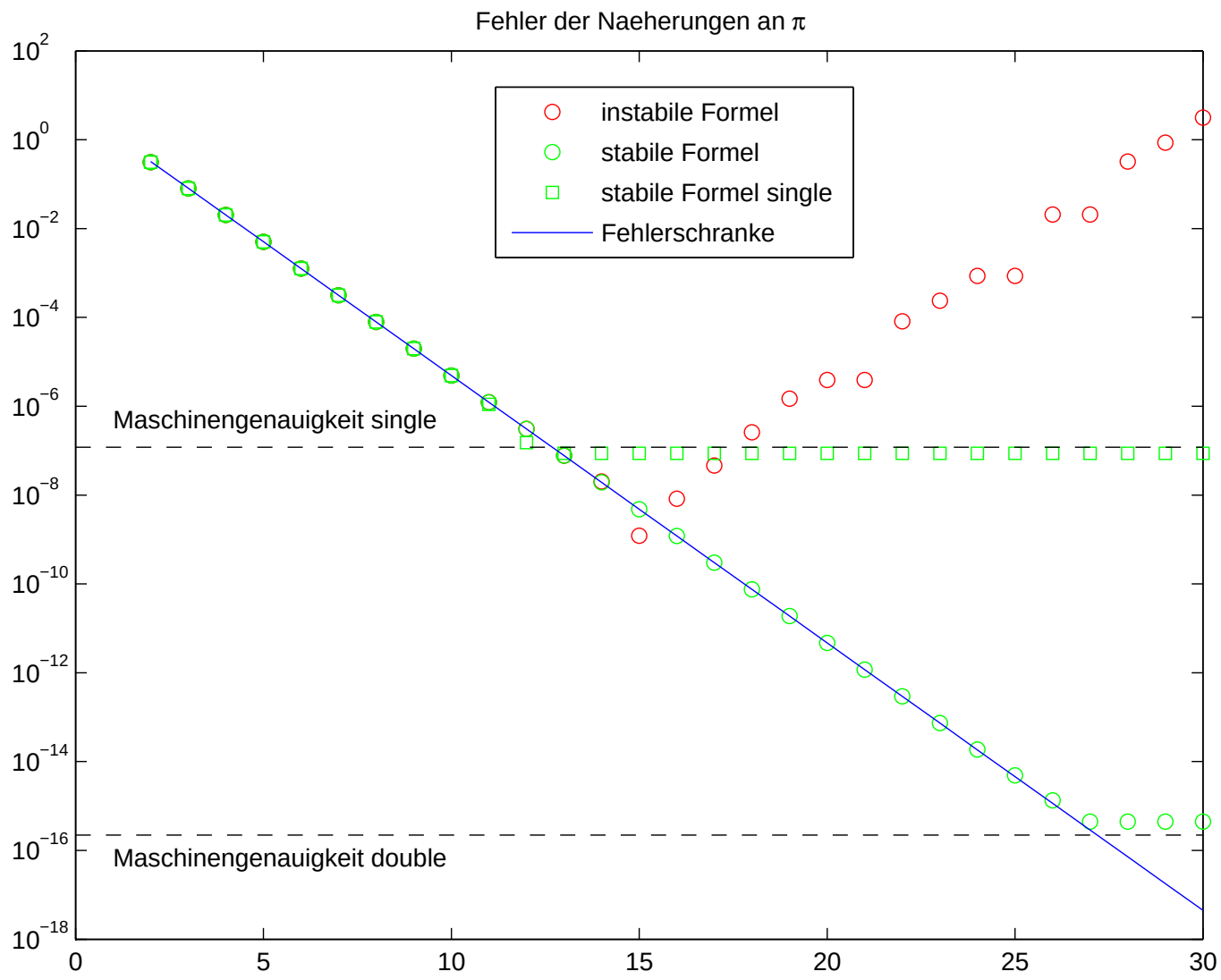
$$R_n := 4 \, \frac{1 - \sqrt{1 - (A_n/2^n)^2}}{2}, \text{ d.h. } A_{n+1} = 2^n \sqrt{R_n}.$$

Beachte: $R_n = 4Z_n$ und Z_n ist (die kleinere) Lösung von

$$X^2 - X + \frac{1}{4} \, (A_n/2^n)^2 = X^2 - X + (A_n/2^{n+1})^2 = 0.$$

Alter Trick:

$$Z_n = \frac{2(A_n/2^{n+1})^2}{1 + \sqrt{1 - (A_n/2^n)^2}}, \; R_n = 4Z_n, \; A_{n+1} = 2^n \sqrt{R_n}.$$


Fehler der Naeherungen an π
instabile Formel
stabile Formel
stabile Formel single
Fehlerschranke
Maschinengenauigkeit single
Maschinengenauigkeit double
10^{2}
10^{0}
10^{-2}
10^{-4}
10^{-6}
10^{-8}
10^{-10}
10^{-12}
10^{-14}
10^{-16}
10^{-18}
0
5
10
15
20
25
30