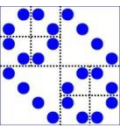


Numerische Programmierung (Konkrete Mathematik)

Literatur

- Numerik für Informatiker (Huckle/Schneider) = Numerische Methoden
- Folien voriger Semester (Vorsicht!)
- Herzberger: Wissenschaftliches Rechnen
- Opfer: Numerik für Anfänger
- Überhuber: Computer-Numerik
- Kahaner/Moler/Nash: Numerical Methods and Software
- Dahmen/Reusken: Numerik für Ingenieure und Naturwiss.



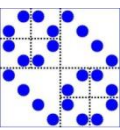
I. Warum Numerik?

1. Numerik ist Teil der Computer Science,
genauso wie Datenbanken, Software Engineering, ...
*Historische Anfänge: Rechnen. Zuse, Manhattan
Project, ...*

Alles was mit Computer zu tun hat, gehört zu CS!

z.B. - Rechner-Arithmetik, Parallelrechner, GPU, usw.

- Computergraphik (**Flächen, Kurven**)
- Bildverarbeitung (Komprimierung, Filtern, Analyse) : **JPEG**
- Audio-verarbeitung: **MP3**
- Information Retrieval, Data Mining, Big Data
- Deep Learning, KI
- Prozessverwaltung, ...



2. Wissenschaftliches Rechnen

Verbund von Natur/Ingenieur-wissenschaft, Mathematik, Numerik, Informatik zur Lösung wiss. Probleme:
z.B. Wettervorhersage, Raketensteuerung, Börse,
Strömungs-Simulation (NASA, BMW, Siemens,...)
Vorhersagen!!! Datenanalyse!!! Steuerung

3. Numerik zur Lösung von Informatik-Problemen

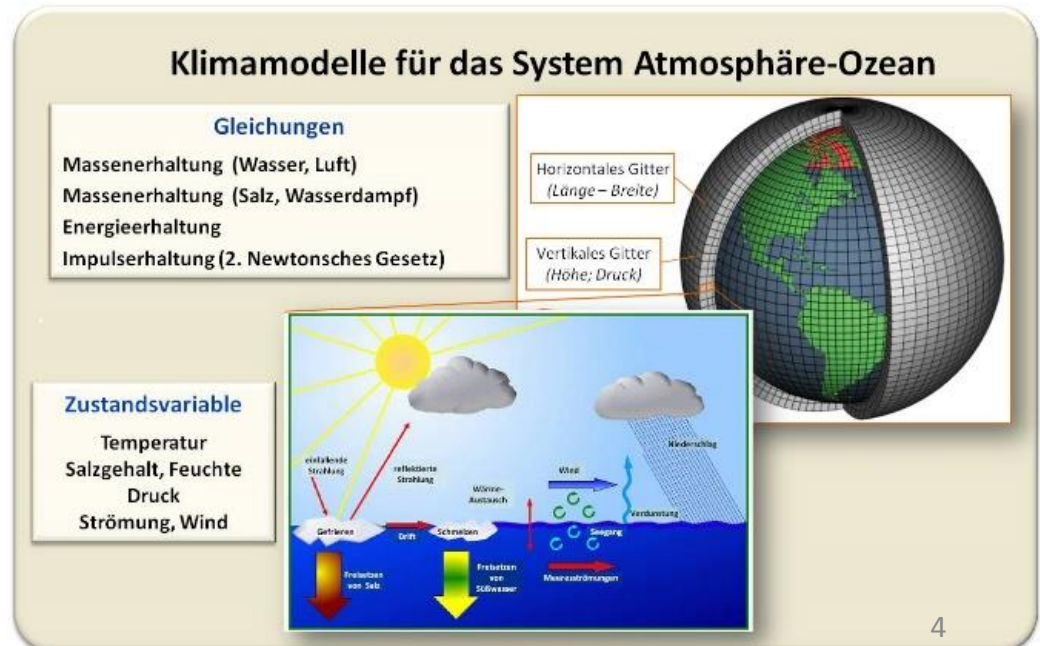
- z.B.
- Warteschlangen, Betriebsmittelzuteilung, und Stochastische Automaten,
 - Räuber-Beute-Modelle
 - Neuronale Netze und Fuzzy-Logik,
 - Machine Learning, Tensorflow
 - Robotics

4. Speziell Games Engineering

- Computergraphik
- Bildverarbeitung
- Realistische Darstellung von Szenen durch Quasi-Lösung physikalischer Gleichungen

Beispiel: Modellierung der Erde:

- Gitter über Kugel
- Werte sammeln
- Physik. Gleichungen



5. Voraussetzungen aus der Informatik:

- Zahldarstellung
- Programmiersprache (JAVA)
- (Komplexitätstheorie)

6. Zur Mathematik:

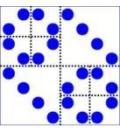
Numerik als Fortführung der ‚Reinen Mathematik‘
mit anderen Mitteln:

z.B. bestimme $\int_0^b e^{-t^2} dt$,

- ^a Gleichungssysteme, Nullstellen
- Grenzwerte, Eigenwerte (Resonanzen)
- Interpolation

Beschreibung der Welt kontinuierlich oder diskret?

Quantenphysik, klassische Physik, Mathematik, Numerik?



Voraussetzungen aus der Mathematik:

Taylor-Entwicklung und Mittelwertsatz, Ableitungen.
Summen und Reihen, trigonometrische Funktionen,
Exp-Funktion, Polynome.

Lineare Gleichungssysteme, Vektoren und Matrizen.
Normen.

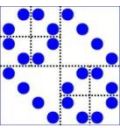
Komplexe Zahlen.

Falls Fragen zu mathematischen Grundlagen auftauchen:
Bitte sofort klären!

Ableitung von x^k , e^x , $\sin(x)$?

$$\begin{pmatrix} 1 & 1 \end{pmatrix} \begin{pmatrix} 2 & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ -1 \end{pmatrix} = ?$$

$$\begin{pmatrix} 2 & 1 \\ 0 & 1 \end{pmatrix} x = \begin{pmatrix} 1 \\ -1 \end{pmatrix}; x = ?$$



II. Rechnerarithmetik und Rundungsfehler

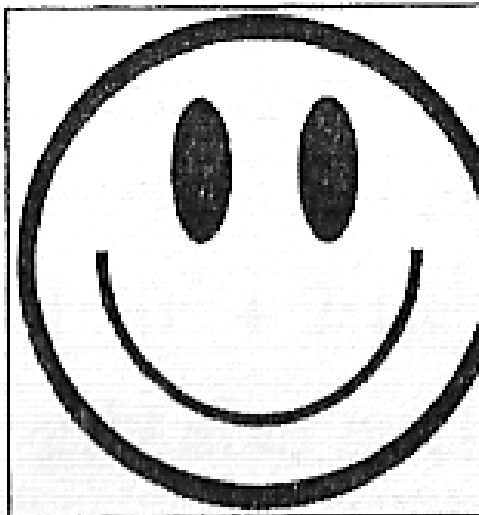
Motto:

„Die natürlichen Zahlen hat der liebe Gott gemacht, alles andere ist Menschenwerk.“

L. Kronecker

“Nobody is perfect”

Osgood zu Daphne (Jerry) in ‘Some like it hot’



0	0	1	1	1	1	1	0	0	0
0	1	0	0	0	0	0	1	0	0
1	0	0	1	0	1	0	0	1	0
0	0	0	1	0	1	0	0	0	1
0	0	0	1	0	1	0	0	0	1
0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	1	0	1
0	0	1	0	0	0	1	0	0	1
1	0	0	1	1	1	0	0	1	0
0	1	0	0	0	0	0	1	0	0
0	0	1	1	1	1	1	0	0	0

Problem: Endlichkeit!

Die Menge **IR** der reellen Zahlen besteht aus unendlich vielen Zahlen mit unendlich vielen Stellen! $\mathbb{N} \rightarrow \mathbb{Z} \rightarrow \mathbb{Q} \rightarrow \mathbb{R}$

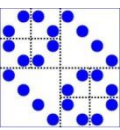
Jeder Computer ist endlich!

2.1. Definition:

Die endliche Menge **M** der in einem Rechner darstellbaren Zahlen heißt

Maschinenzahlen

Wir unterscheiden ganzzahlige Maschinenzahlen und Maschinenzahlen für reelle Zahlen (=Gleitpunktzahlen).



1.Problem: Abbildung $\mathbb{R} \rightarrow M$ liefert Fehler

2.Problem: Arithmetische Operationen $+ - * / :$
 $M \times M \rightarrow M$ gilt nicht!

*Beispiel: $1.1 * 1.1$ bei Zahlen mit einer Stelle nach dem Komma*

Zum ersten Problem:

2.2 Definition:

Eine Abbildung $rd: \mathbb{R} \rightarrow M$ bezeichnen wir

als Rundung, wenn gilt: $|x - rd(x)| = \min_{m \in M} |x - m|$

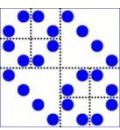
Dies ist keine eindeutige Definition von rd !

Andere Möglichkeiten statt rd : ceil, floor, to zero

Minimalforderung:

$$rd(m) = m \quad \text{für} \quad m \in M \subset \mathbb{R}$$

Rundungsfehler (absoluter): $f_{rd}(x) := x - rd(x)$





Zum zweiten Problem (arithmetische Operationen) :

$$+_M : M \times M \mapsto M$$

$$x +_M y := rd(x + y)$$

Genauso definieren wir für
die Näherungsoperatoren

$-$, $*$, $/$

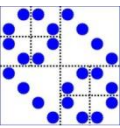
$-_M$, $*_M$, $/_M$

Beispiele von Maschinenzahlen:

Quantisierung von Musik- oder Bildwerten: 8 Bit, 16 Bit
Geldbeträge, Wechsel/Aktienkurse: DAX 3577,72

Alternativen zu Maschinenzahlen und -arithmetik?

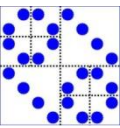
- Rechnen mit beliebig vielen Stellen
(symbolisches Rechnen, MATHEMATICA, MAPLE)
- Rechnen mit Intervallen (Intervall-Arithmetik)



Folien-Beispiel: $0.1114 + 0.001116$

Modell: Jede Zahl repräsentiert durch drei
Dezimalstellen und Position des Dezimalpunkts:

*Volle Stellenzahl führt zu aufwendigen Rechnungen,
Zahlen würden zu lang!
Speicherplatz!
Rechenzeit!
Meist keine nützliche Information!*



Intervallarithmetik auf 3 Stellen (Folien-Beispiel):

Repräsentiere Zahlen durch Einschließungsintervalle:

$$0.1114 \in [0.111 \quad 0.112]$$

$$0.001116 \in [0.00111 \quad 0.00112]$$

Untere Intervallgrenze der Lösung: $\lfloor 0.111 + 0.00111 \rfloor = \lfloor 0.11211 \rfloor = 0.112$

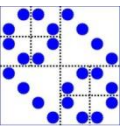
Obere Grenze: $\lceil 0.112 + 0.00112 \rceil = \lceil 0.11312 \rceil = 0.114$

Resultat: Lösung von $0.1114 + 0.001116$ liegt
im Intervall $[0.112, 0.114]$

Vorteil: Exakte Information über Lage und Qualität der Lösung

Nachteil: Rechenzeit! Ev. Ergebnisintervall sehr groß!

Maschinenzahlen: Paar $[x, y]$, $x, y \in M$.



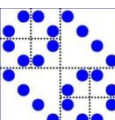
2.3. Definition: Festkommazahlen als Maschinenzahlen

$n_1 n_2 \cdots n_k, m_1 \cdots m_j$ mit

$n_1, n_2, \dots, n_k, m_1, \dots, m_j \in \{0, 1, \dots, 9\}, \quad k, j \in \mathbb{N}$

Beispiele:

- Geldbeträge wie Euro.Cent , $j=2$
- Wechselkursangaben, wie 1.1591 (\$ zu €), $j=4, k=1$
- Börsenindizes: DAX 3577.72, $j=2, k=4$.
- Alte Taschenrechner.





Für praktisches Rechnen ungenügend!

Zu wenige ganze, bzw. reelle Zahlen darstellbar!

Sehr große Zahlen? Sehr kleine Zahlen? Zahlen mit vielen Nachkommastellen? Zusätzlich große Rundungsfehler:

Börsenindex Vancouver 1983:

Start des Indexes mit Wert 1000.

Bei jedem Verkaufereignis (ca. 3000 pro Tag) wurde der Index neu berechnet auf drei Stellen nach dem Komma:

Rechne mit vier Stellen nach dem Komma und dann Abschneiden der vierten Stelle.

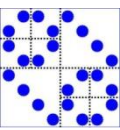
(Quasi rechnen wie mit ganzen Zahlen in C)

Frage: Wo lag der Index nach 2 Boom-Jahren?

Nach 22 Monaten wurde 574.081 angegeben.

Der ‚wahre‘ Wert: 1098.892

Systematischer Fehler, der sehr oft auftritt! Update!



Landtagswahl 1992 in Schleswig-Holstein:

Grüne erhielten 4.97%.

Zur Darstellung der Ergebnisse Festkomma mit einer Stelle nach dem Komma, (j=1);

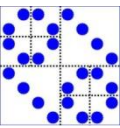
also Anzeige des Ergebnisses als 5.0%.

Fehler wurde erst entdeckt, nachdem offizielle Ergebnisse bereits veröffentlicht waren.

Rechnen(?) mit EXCEL!!!

Diese Zahlendarstellungen werden wir i.F. nicht verwenden

Lance Armstrong Bug, Heisenbug, Bohrbug,...



2.4. Integer(Maschinen)zahlen

Endlicher Ausschnitt aus den ganzen Zahlen \mathbf{Z}
symmetrisch um Null angeordnet.

Man verschiebt den Ausschnitt so, dass alle Zahlen positiv
werden: Warum? $-n, \dots, -1, 0, 1, \dots, n \rightarrow 0, 1, \dots, m-1, m=2n$

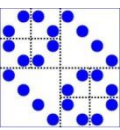
Menge der Integer-Maschinenzahlen M

$$z = \sum_{i=0}^{t-1} m_i \times 2^i - 2^{t-1}, \quad m_i \in \{0,1\}, \quad i = 0, \dots, t-1$$

$t \in \mathbb{N}$ gibt die Stellenzahl an, bzw. Bits.

Ursprüngliches z wird als nichtnegative Maschinenzahl
gespeichert und repräsentiert durch die m_i .

Frage: Welchem z entspricht $m=0$? Welches m entspricht $z=0$?
Minimales/maximales z ?



Stufenzahlen: 2^i mit $0 \leq i \leq t-1$

Also $M = \langle -2^{t-1}, 2^{t-1}-1 \rangle$

Bei 32 Bits (=4 Bytes) ergibt sich der Zahlenbereich der ganzzahligen Maschinenzahlen daher zwischen

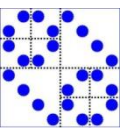
$$-2^{31} = -2147483648 \quad \text{und} \quad 2^{31}-1=2147483647$$

Beispiel:

Darstellung der Dezimalzahl 11 mit $t = 5$

$(11011)_2$ oder $(01011)_2$?

$$\begin{aligned}
 11 &= 27 - 16 && \text{(da } 11+16=27\text{)} \\
 &= (16 + 11) - 16 = 16 + (8+3) - 2^4 \\
 &= 16 + 8 + (2+1) - 2^4 \\
 &= 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 - 2^4 \\
 &\rightarrow (11011)_2, && \text{nicht } 11=(01011)_2 \\
 \text{oder} & \quad -5 = 11 - 16 \rightarrow (01011)_2
 \end{aligned}$$



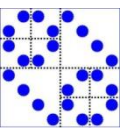


Zu Beachten:

- Fehler bei Bereichsüberschreitung (Overflow)
ev. *Wrap-around-Effekt. (Therac-25)*
- Integer-Division in Programmiersprachen: $1 / 3 = 0$
als Division ohne Rest oder Rundung zur Null
(Abschneiden).
- Division durch 0 (*Beispiel: USS Yorktown*)

Vorteil: Null hat nur eine Darstellung!

*Sonst treten bei Integerzahlen keine Rundungsfehler auf!
Also muss man nur beachten, dass man genügend viele
Bit zur Verfügung hat, um die gewünschten Zahlen
darstellen zu können.*



2.5. Gleitpunktzahlen

Stelle $x \in \mathbb{R}$ durch Vorzeichen, Mantisse m und Exponenten e dar, bzgl. Basis $b > 1$:

$$x = (-1)^v \cdot m \cdot b^e$$

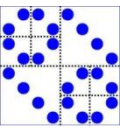
Wir betrachten nur $b=2$ ($b=8, 10, 16$ kommen kaum vor)

Beispiel:

Darstellungen der Zahl 16:

$$\dots = 0.25 \cdot 2^6 = 0.5 \cdot 2^5 = 1 \cdot 2^4 = 2 \cdot 2^3 = 4 \cdot 2^2 = \dots$$

mit $v=0$, $m = 2^{-i}$ und $e = i+4$.

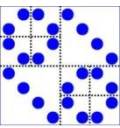


**Normierung ist notwendig, damit Darstellung eindeutig:
Der Exponent ist stets so zu wählen, dass die
Mantisse m genau eine von Null verschiedene Stelle
vor dem Komma hat.**

Also in unserem Beispiel $16 = + 1.0 \cdot 2^4$

Vorsicht: andere Bücher normieren so, dass erste Stelle
hinter dem Komma von Null verschieden!
(also $16 = + 0.5 \cdot 2^5 = + (0.1)_2 \cdot 2^5$)

Weiterer Vorteil: Normierte erste 1 muss gar nicht
gespeichert werden, ist gesetzt!



2.6. Definition: Normierte Gleitpunktzahlen

Die Menge M der (reellen) Maschinenzahlen besteht aus Zahlen der Form

$$x = (-1)^v \cdot \left(\sum_{i=0}^{t-1} x_i 2^{-i} \right) \cdot 2^e$$

Vorzeichen: $v \in \{0,1\}$,

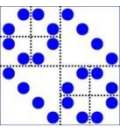
Mantisse: $x_0 = 1, \quad x_i \in \{0,1\} \quad \text{sonst,}$

Exponent: Integerzahl e .

Also hat die Mantisse eigentlich t Stellen;
aber die erste Stelle muss nicht gespeichert werden,
da sie wegen der Normierung 1 ist.

Daher werden für die Mantisse nur $(t-1)$ -Bits gebraucht!

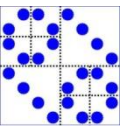
Problem dabei?



Bei Zahlen nahe 0 kann die Normierung aufgehoben werden (sog. subnormale Gleitpunktzahlen)!

Ausnahmeregeln, falls Exponent minimal oder maximal ist!

Exponent wird gespeichert als ganzzahlige Integer-Maschinenzahl wie vorher beschrieben!



Beispiel:

Stufenzahlen: $2^3, 2^2, 2^1, 2^0, 2^{-1}, 2^{-2}, 2^{-3}, 2^{-4}, \dots$

13.6=?

$$= 8 + 5.6$$

$$= 8 + (4+1.6)$$

$$= 8 + 4 + (1+0.6)$$

$$= 8 + 4 + 1 + (1/2 + 0.1)$$

$$= 8+4+1+1/2 + (0.0625 + 0.0375)$$

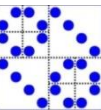
$$= 8+4+1+1/2+1/16 + (0.03125+0.00625)$$

Also im Zweiersystem

$$13.6 = (1101.10011\dots)_2$$

Als normierte Gleitpunktzahl

$$13.6 = (-1)^0 \cdot (1.10110011\dots)_2 \cdot 2^3 \quad \text{mit } e=3.$$

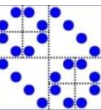


Die Zahl 13.6 in verschiedenen Genauigkeiten:

$$13.6 = (1.10110011\dots)_2 * 2^3$$

t	<i>Darstellung</i>	<i>Rundung</i>	<i>Fehler</i>
2	$(1.1)_2 * 2^3$	<i>ab</i>	$(1.6)_{10}$
3	$(1.11)_2 * 2^3$	<i>auf</i>	$-(0.4)_{10}$
4	$(1.110)_2 * 2^3$	<i>auf</i>	$-(0.4)_{10}$
5	$(1.1011)_2 * 2^3$	<i>ab</i>	$(0.1)_{10}$
6	$(1.10110)_2 * 2^3$	<i>ab</i>	$(0.1)_{10}$
7	$(1.101101)_2 * 2^3$	<i>auf</i>	$-(0.025)_{10}$

$(1.1)_2 2^3 = 8+4=12$; $(1.11)_2 2^3 = 8+4+2=14$; $(1.1011)_2 2^3 = 8+4+1+0.5=13.5$;



Anzahl der für Mantisse und Exponent benutzten Bit und daraus sich ergebende Rechengenauigkeit und Exponentenbereich bei IEEE-Datentyp float und double:

<i>Typ</i>	<i>Mantisse</i>	<i>t</i>	<i>Exponent</i>	$[e_{\min}, e_{\max}]$	ε
<i>float</i>	$23_{(+1)}$		8	$[-126, 127]$	$2^{-24} \approx 6 * 10^{-8}$
<i>double</i>	$52_{(+1)}$		11	$[-1022, 1023]$	$2^{-53} \approx 1 * 10^{-16}$

↑
Relativer Fehler bei Rundung



IEEE-Standard (single precision, 32 Bit):



Exponent e mit 8 Bit, gespeichert in der ‚positiven‘ Form

$$p = e + 127 \quad (\text{also Shift } 2^{t-1}-1, \text{ nicht } 2^{t-1}).$$

$p=(0000\ 0000)_2$ ist dann kleinstmöglicher Exponent.

Ist auch noch in der Mantisse (bis auf Normierung)

alles 0, so wird diese Zahl als Null interpretiert.

$$p=(0000\ 0001)_2 \rightarrow e=-126, \quad p=(1111\ 1110)_2 \rightarrow e=127$$

Entsprechend $p=(1111\ 1111)_2 = 255$

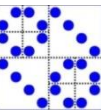
als unendlich (NaN oder Not a number)

$$-126 \leq e \leq 127 \quad \text{entspricht} \quad 1 \leq p \leq 254$$

Für die Mantisse bleiben 23 Bit

(24 unter Berücksichtigung der Normierung).

1 Bit für Vorzeichen \rightarrow Insgesamt $1+23+8=32$ Bit .



Beispiel Ariane 5



1996 endete die erste Ariane 5 durch Selbstzerstörung
40 sec nach Start.

Ursache:

36.7 sec nach Start versuchte der Bordcomputer den
Wert der horizontalen Geschwindigkeit von 64 Bit
Gleitpunkt in 16 Bit signed Integer umzurechnen.

Der sich ergebende Wert war zu groß →

- Overflow
- Absturz des Computers,
- Übergabe an Back-up-Rechner, der aber aus demselben Grund bereits abgestürzt war
- Kein Lenksystem mehr
- instabiler Flug
- Selbstzerstörung.

Benutzte Software stammte von Ariane 4.

Ariane 5 war schneller!

Umwandlung war nicht abgesichert!

Historisch: Konrad Zuse verwendete in seinen Z1-Z4 bereits Gleitpunktzahlen.

Rundung

2.7. Def.: Für $x = (-1)^v \cdot 2^e \cdot 1.x_1x_2\dots x_{t-1}x_t\dots$ definieren wir

$$rd(x) = (-1)^v \cdot 2^e \cdot 1.x_1x_2\dots x_{t-1} \text{ für } x_t = 0$$

$$rd(x) = (-1)^v \cdot 2^e \cdot 1.x_1x_2\dots x_{t-1} + 2^{-t+1} \text{ für } x_t = 1 \text{ und } x_tx_{t+1}x_{t+2}\dots \neq 1000\dots$$

$$rd(x) = (-1)^v \cdot 2^e \cdot 1.x_1x_2\dots x_{t-1} \text{ für } x_t = 1 \text{ und } x_{t-1}x_tx_{t+1}\dots = 0100\dots$$

$$rd(x) = (-1)^v \cdot 2^e \cdot 1.x_1x_2\dots x_{t-1} + 2^{-t+1} \text{ für } x_t = 1 \text{ und } x_{t-1}x_tx_{t+1}\dots = 1100\dots$$

1. Fall: Abschneiden, falls $x_t = 0$;
2. Fall: Letztes Bit wird um eins erhöht, falls
 $x_t = 1$ und $x_tx_{t+1}x_{t+2}\dots \neq 1000\dots$
3. Fall: ($x_t = 1$) $rd(1.x_1x_2\dots x_{t-2}0|1) = 1.x_1x_2\dots x_{t-2}0$
 oder $rd(1.x_1x_2\dots x_{t-2}1|1) = 1.x_1x_2\dots x_{t-2}1 + 2^{-t+1}$

Rundung so, dass letztes Bit 0 wird.

Im Dezimalsystem: $11.5 \rightarrow 12$, aber $10.5 \rightarrow 10$

*Es kann ev. Overflow auftreten (zu großer Exponent).
Dieser Fall wird aber im Folgenden ignoriert
(Fehlermeldung?)*

*Es kann Underflow auftreten.
In der Regel wird dann einfach die kleine Zahl zu 0 gesetzt.*

Die Rundungsfehleranalyse in den folgenden Abschnitten
wird nur für Normalfall durchgeführt (ohne Over/Underflow)
Was ist wichtiger für Fehlerbetrachtung:
Mantisse oder Exponent?

Nur die Mantisse ist wichtig!
Der Exponent spielt keine Rolle, weil dabei praktisch keine
Fehler auftreten!

2.8. Absoluter Rundungsfehler:

$$f_{rd}(x) = x - rd(x)$$

$$|f_{rd}(x)| = |x - rd(x)| \leq \left(\frac{1}{2} 2^{-t+1} \right) 2^e = 2^{e-t}$$

Problem: Ein absoluter Fehler von der Größe 0.1 ist

- bei der Zahl 2.1 recht groß, aber
- bei der Zahl 123456.7 sehr klein.

Beispiel: 1 Million + 1 Jahr alter Dinosaurierknochen
 Newton: Verhältnis Erde:Mond = 39.371 (81)
 Geschwindigkeit, mit der Mond auf
 Erde fällt: 14.7706353 Fuß/min
 Vorspiegelung hoher Genauigkeit!

Sinnvollere Definition des Rundungsfehlers?



$$f_{rel}(x) := \varepsilon_x := \frac{f_{rd}(x) - x}{x} = \frac{\delta_x}{x} = \frac{x - rd(x)}{x} \quad \text{für } x \neq 0$$

$$\text{Dann gilt: } |f_{rel}(x)| \leq \frac{2^{e-t}}{|x|} \leq \frac{2^{e-t}}{2^e} = 2^{-t}$$

da wegen Normierung die Mantisse in $[1,2]$ liegt:

Mantisse stets $1 \leq (1.\text{bbbbbb}\dots)_2 < 2$

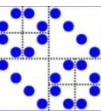
also $m \geq 1$, d.h. $|x| \geq 2^e$

Außerdem gilt durch Umformung von 2.9:

$$\mathbf{2.10.:} \quad rd(x) = x - x\varepsilon_x = x(1 - \varepsilon_x)$$

$$\text{mit } |\varepsilon_x| \leq 2^{-t}$$

Gerundete Zahl=Ausgangszahl, bis auf Faktor $(1 \pm \varepsilon)$



Def.: Die obere Schranke für den relativen Fehler, der bei der Rundung mit t-stelliger Mantisse auftreten kann, heißt **Maschinengenauigkeit** ε , und ergibt sich als

$$\varepsilon = 2^{-t}$$

Andere Möglichkeit, die Maschinengenauigkeit zu definieren:
Größte positive Zahl $y=2^{-k}$, so dass

$$1.0 + y = 1.0$$

Beispiel $t=2$, $\varepsilon = \frac{1}{4} = (0.01)_2$; $(1.0|1)_2 \rightarrow (1.0)$

Mantissenlänge (Bits) \leftrightarrow Genauigkeit

Zusammenfassung

Relativer Rundungsfehler: $|f_{rel}(x)| := \left| \frac{f_{rd}(x)}{x} \right| = \left| \frac{x - rd(x)}{x} \right| \leq 2^{-t} = \varepsilon$

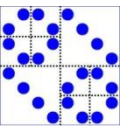
Maschinengenauigkeit ϵ , Mantissenlänge t , durch Normierung erstes Bit gleich 1.

Was könnte problematisch sein?

Für jede sauber implementierte Operation/Funktion $op(x)$ gilt :

$$op_M(x) = rd(op(x)) = op(x)(1 + \varepsilon'), \quad |\varepsilon'| \leq \varepsilon$$

Beispiel: Stelle die Zahl $1/10$ binär dar und runde auf 3 Mantissen-Stellen.



MATLAB und Heisenberg Effekt

```
function x = myrealmin()
x = 1;
temp = x;
while eps * temp / 2 > 0
    temp = (eps * temp / 2) ;
    if (temp > 0)
        x = temp;
    end
end
```

heisen.m

```
a = 1;
while a*eps>0
    last = a;
    a = a/2.0;
end
a = last
```

Verschiedene Ergebnisse abhängig von Ausgabe oder Script.

Aktuell: NVIDIA Titan V GPU

