Bericht zu tools4.py und ab4.py

theoretische und praktische Untersuchung von Maschinenzahlen

T. Hadamczik und T. Höfting

31. Mai 2019

Inhaltsverzeichnis

1 Einführung in Theorie

1.1 Zahlendarstellung

Da jeder Computer nur endliche Speicher und Rechenkapazitäten besitzt, ist es nicht möglich die reellen Zahlen vollständig darzustellen bzw. mit ihnen zu rechnen, da diese ja bekanntlich unendlich lange Ziffernfolgen seinen können. Nach dem Darstellungssatz für reelle Zahlen [**?**] gibt es zu jeder Zahl $x \in \mathbb{R} \setminus \{0\}$ eine Darstellung der Gestalt

$$x = (-1)^v \beta^N \sum_{i=1}^{\infty} x_i \beta^{-i} \tag{1.1}$$

mit einem beliebigen $\beta \in \mathbb{N}$, $\beta \geq 2$, $v \in \{0,1\}$, $N \in \mathbb{Z}$ und $x_i \in \{0, 1, \ldots, \beta - 1\}$, wenn zusätzlich für die Zahlen x_i gilt, dass $x_1 \neq 0$ und dass zu jedem $n \in \mathbb{N}$ ein Index $i \geq n$ existiert mit $x_i \neq \beta - 1$. Wir betrachten $\beta = 2$, d. h. wir schauen uns das Binärsystem an, in dem die meisten Rechner arbeiten. Allerdings haben wir jetzt immer noch das Problem, dass ein Computer nicht unendlich viele x_i speichern kann. Wir sagen x ist eine *normalisierte Gleitkommazahl*, falls

$$x = (-1)^v 2^N \sum_{i=1}^{t} x_i 2^{-i} \tag{1.2}$$

mit $v \in \{0,1\}$, $N_{min} < N \leq N_{max}$ und $x_i \in \{0,1\}$ $\forall$ $i = 1,\ldots,t$, $x_1 = 1$. t ist die *Mantissenlänge*. Die Menge der *normalisierten Gleitkommazahlen* und der *subnormalen Gleitkommazahlen* bilden zusammen die Menge der *Maschinenzahlen*

1.2 Runden

Durch Runden können wir jedes $x \in \mathbb{R}$ auf die nächstgelegene Maschinenzahl abbilden. Dafür haben wir folgende Rundungsvorschriften:
Sei $t \in \mathbb{N}$ die *Mantissenlänge* und $\mathrm{x} \in \mathbb{R}_N \cup \mathbb{R}_D \setminus \{0\}$ besitze die normalisiert Gleitkommadarstellung $x = (-1)^v 2^N \sum\limits_{i=1}^{\infty} x_i 2^{-i}$. Dann gilt

- für $N_{min} < N \leq N_{max}$: $\mathrm{rd}_t(x) := \begin{cases} (-1)^v 2^N \sum\limits_{i=1}^{t} x_i 2^{-i} & \text{für } x_{t+1} = 0 \\ (-1)^v 2^N (\sum\limits_{i=1}^{t} x_i 2^{-i} + 2^{-t}) & \text{für } x_{t+1} = 1 \end{cases}$

- für $N \leq N_{min}$- $t : \mathrm{rd}_t(x) := 0$
- für N_{min}- $t < N \leq N_{min}$: $\mathrm{rd}_t(x) := \begin{cases} (-1)^v 2^{N_{min}} \sum_{j=n+1}^{t} x_{j-n} 2^{-j} & \text{für } x_{t+1-n} = 0 \\ (-1)^v 2^{N_{min}} \sum_{j=n+1}^{t} x_{j-n} 2^{-j} + 2^{-t} & \text{für } x_{t+1-n} = 1 \end{cases}$

  wobei $n := N_{min} \ - N$.

1.2.1 Rundungsfehler

Die wichtigsten Abschätzungen, die man macht sind der relative Fehler, und der absolute Fehler. Es gilt für den absoluten Fehler

$$\delta(x) := |rd_t(x) - x| \leq 2^{N-t-1} \tag{1.3}$$

und für den relativen Fehler

$$\epsilon(x) := |\frac{\delta(x)}{x}| \leq 2^{-t}. \tag{1.4}$$

Daraus folgt, dass die *relative Rechengenauigkeit* $\tau := 2^{-t}$ ist.

2 Schnittstellendokumentation des Python-Programms

2.1 tools4.py

Die Datei stellt die Funktionen zur Verfügung, die wir für unsere Experimente brauchen. Zum einen sind dies zwei Funktionen, die die relative Maschinengenauigkeit berechnen. Desweiteren wird eine Funktion definiert, die die beiden Seiten der Formel

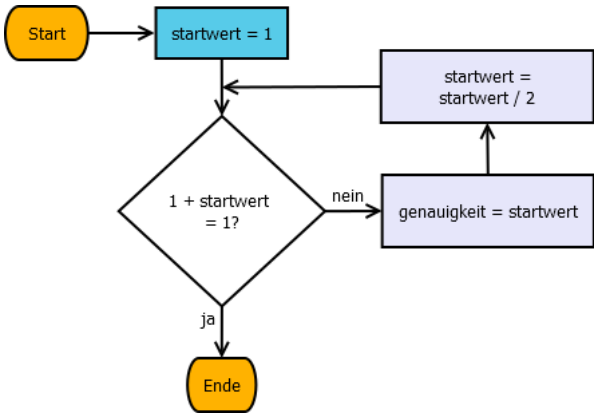
$$\frac{1}{x} + \frac{1}{x+1} = \frac{1}{x(x+1)} \tag{2.1}$$

berechnen. `tools4.py` besitzt kein Hauptprogramm. Benötigt wird die Klasse *Decimal* aus dem Paket *decimal*, `numpy` und *matplotlib*.

2.1.1 Funktionen

`accuracy1(datatype)`

Diese Funktion berechnet die relative Maschinengenauigkeit in Abhängigkeit vom übergebenen Datentyp. Die Genauigkeit wird entsprechend des Algorithmus bestimmt, welcher im folgenden Flussbilddiagramm abgebildet ist.



Input:

`datatype` *(str)* Der Datentyp, für den die relative Maschinengenauigkeit berechnet werden soll. Einer der Werte numpy.float64, numpy.float32 oder numpy.float16.

Return:

`genauigkeit` *(numpy.float)* Die relative Maschinengenauigkeit, wobei der Datentyp der Ausgabe dem als Parameter übergebenen Datentyp entspricht.

`accuracy2(datatype)`

Diese Funktion berechnet die relative Maschinengenauigkeit in Abhängigkeit vom übergebenen Datentyp. Die Genaugkeit wird bestimmt, indem der Term $\frac{7}{3} - \frac{4}{3} - 1$ berechnet wird, wobei die Datentypen der Zahlen jeweils dem übergebenen Datentyp entsprechen.

Input:

`datatype` *(str)* Der Datentyp, für den die relative Maschinengenauigkeit berechnet werden soll. Einer der Werte numpy.float64, numpy.float32 oder numpy.float16.

Return:

`genauigkeit` *(numpy.float)* Die relative Maschinengenauigkeit, wobei der Datentyp der Ausgabe dem als Parameter übergebenen Datentyp entspricht.

`experiment(wert, laenge)`

Die Funktion berechnet die linke (:= yl) und die rechte (:= yr) Seite der Gleichung im Datentyp *decimal.Decimal*, sowie die absolute und relative Abweichung zu einer gegebenen Mantissenlänge und einem gegebenen x.

Input:

`wert` *(float)* der x-Wert

`laenge` *(int)* die Mantissenlänge der Maschinenzahlen

Return:

`yl` *(decimal.Decimal)* das Ergebniss der linken Seite der Gleichung

`yr` *(decimal.Decimal)* das Ergebniss der rechten Seite der Gleichung

`fehler_absolut` *(decimal.Decimal)* die Differenz der beiden Seiten (absoluter Fehler)

`fehler_relativ` *(decimal.Decimal)* die relative Abweichung der beiden Seiten (relativer Fehler)

`plotten()`

Diese Funktion plottet ein Diagramm, dass den relativen Fehler in Verhältnis zur Mantissenlänge setzt.

Input:

`x1` *(float)* eine Ganzzahl.

`x2` *(float)* eine Ganzzahl, die ungleich 0 sein muss.

`x3` *(float)* eine Ganzzahl, die ungleich 0 sein muss.

Return:

–

2.2 a4.py

Das Programm dient zur Durchführung der Experimente und importiert hierzu alle Funktionen aus *tools4*.

2.2.1 Hauptprogramm

Die `main()`-Funktion ist unser Hauptprogramm. In ihm können die Experimente ausgeführt werden. Die Eingabedaten, die für die Experimente gebraucht werden, werden aus der Datei `daten.txt` eingelesen. Die Daten werden zeilenweise eingelesen (jede Zeile wird unter einer Variablen abgelegt) und müssen folgende Reihenfolge haben:

```
<datentyp> --- für die Funktion accuracy1()
<x1>
<x2>
<x3> --- x-Werte um mit dem relativen Fehler zu experimentieren
<laenge> --- Mantissenlaenge, die die Decimal Instanzen haben sollen
```

Unter Verwendung der eingelesenen Daten wird dann unter Benutzung der Funktion *accuracy1()* die relative Maschinengenauigkeit für den unter Datentyp abgespeicherten Datentyp bestimmt und ausgegeben. Darüber hinaus wird unter Verwendung der Funktion *experiment()* der relative Fehler für die eingelesenen x-Werte in Abhängigkeit von der eingelesenen Mantissenlänge bestimmt und ausgegeben. Schließlich wird unter Verwendung der Funktion *plotten()* ein Diagramm erstellt, welches den relativen Fehler für als Funktion von der Mantissenlänge (für die Werte 1-25) für die drei verschiedenen x-Werte darstellt.

3 Ergebnisse von Aufgabe 1

Gegeben ist eine 8-stellige Gleitkommaarithmetik mit $N_{min} = -5$, $N_{max} = 8$.

a)

Bestimmung der größten normalisierten Gleitkommazahl:

Bestimmung der betragsmäßig kleinsten normalisierten Gleitkommazahl:

$$\begin{aligned} x_{min} &= (0.10000000)_2 \cdot 2^{N_{min}+1} = (0.1)_2 \cdot 2^{N_{min}+1} \\ &= (0.1)_2 \cdot 2^{-4} = 1 \cdot 2^{-3} = \frac{1}{16} = 0.0625 = 6.25e-2 \end{aligned}$$

Bestimmung der kleinsten positiven darstellbaren Maschinenzahl:

$$\begin{aligned} x &= 2^{N_{min}} \cdot (0.x_2x_3...x_8)_2 = 2^{-5} \cdot (0.0000001)_2 \\ &= 2^{-5} \cdot 2^{-7} = 2^{-12} = 2,4414062e-4 \end{aligned}$$

Bestimmmung der Abschätzung für den absoluten und relativen Rundungsfehler für reelle Zahlen in R_N :

absoluter Fehler: $|rd_s(x) - x| \leq 2^{N-t-1} = 2^{N-9}$

relativer Fehler: $|\frac{rd_s(x)-x}{x}| \leq 2^{-8}$

b)

Im Dezimalsystem sind die Zahlen $z_1 = 67.0$, $z_2 = 287.0$, $z_3 = 10.625$, $z_4 = 1.01$, $z_5 = 0.0002$ und $z_6 = \frac{1}{3}$ gegeben.

i) Bestimmung der Darstellungen als Dualzahl auf 10 Nachkommastellen genau

$$\begin{aligned} z_1 &= 67 = 1 \cdot 2^6 + 1 \cdot 2^1 + 1 \cdot 2^0 \\ &= (1000011)_2 = (0.1000011) \cdot 2^6 \in M \end{aligned}$$

$$\begin{aligned} z_2 &= 287.0 = 1 \cdot 2^8 + 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 \\ &= (1.00011111)_2 = (0.100011111)_2 \cdot 2^9 \in \mathbb{R}_{overflow} \end{aligned}$$

$$\begin{aligned} z_3 &= 10.625 = 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^{-1} + 1 \cdot 2^{-3} \\ &= (0.1100101)_2 \cdot 2^4 \in M \end{aligned}$$

$$
\begin{aligned}
z_4 = 1.01 &= 1 \cdot 2^0 + 12^{-7} + 0,0021875 = 1 \cdot 2^0 + 12^{-7} + 2^{-9} + 0,000234375 \\
&\approx \cdot 2^0 + 12^{-7} + 2^{-9} + 1 \cdot 2^{-13} \approx (1.00000010100001...)_2 \\
&= (0.100000101)_2 \cdot 2^1 \notin M
\end{aligned}
$$

$$
z_5 = 0.0002 \approx 0.0000000000 \cdot 2^0 = 0
$$

Maschinenzahl der Null M=0, e=00.00000. Es existierteine genaue Darstellung der Null.

$$
\begin{aligned}
z_6 = \frac{1}{3} &= \frac{1}{4} + \frac{1}{16} + \frac{1}{64} + \frac{1}{256} + \frac{1}{2024} + ... \\
&= 2^{-2} + 2^{-4} + 2^{-6} + 2^{-8} + 2^{-10} + ... \approx (0.0101010101)_2
\end{aligned}
$$

ii) Angabe der zugeordneten Maschinenzahlen
iii) Angabe der relativen und absoluten Rundungsfehler

4 Beschreibung der Experimente und Beobachtungen

4.1 Experimente zum relativen Fehler

Wir wollen herausfinden, ob für die beiden analytisch gleichen Terme auf beiden Seiten der Formel (2.1) auch das gleiche herauskommt, wenn wir dies mit dem Computer berechnen, und wenn die beiden Seiten sich unterscheiden, wie groß die Abweichungen sind. Dazu benutzen wir unsere Funktion `experiment()`, die uns für den von uns untersuchten x-Wert und einer gewählten Mantissenlänge unseren relativen und absoluten Fehler liefert. Im ersten Experiment lassen wir die Mantissenlänge t auf 10 und schauen inwiefern sich die Fehler für verschiedene x-Werte unterscheiden. Die folgende Tabelle zeigt die Ergebnisse für x-Werte in verschiedenen Größenordnungen von x:

x	relativer Fehler	absoluter Fehler
1	0.0	0.0
10	$1.0{\cdot}10^{-10}$	$1.0{\cdot}10^{-12}$
10^5	$1.0{\cdot}10^{-10}$	$1.0{\cdot}10^{-20}$
10^6	$1.0{\cdot}10^{-6}$	$1.0{\cdot}10^{-20}$
10^7	$1.0{\cdot}10^{-7}$	$1.0{\cdot}10^{-22}$
10^8	$1.0{\cdot}10^{-8}$	$1.0{\cdot}10^{-26}$
10^9	$1.0{\cdot}10^{-9}$	$1.0{\cdot}10^{-27}$
10^{10}	1.0	10^{-20}
10^{15}	1.0	$1.0{\cdot}10^{-30}$

Anmerkung: In der Tabelle sind Näherungswerte für die Fehler, da nur die Größenordnung interessant ist.
Wir stellen fest, dass der relative und der absolute Fehler kleiner werden für größere x-Werte, bis auf die Ergebnisse für den Wert 10^{10} und 10^{15}. Dieser Sprung, dass der relative Fehler auf einmal 1 ist, ist damit zu erklären, dass für x- Werte, die eine bestimmte Größe haben sich die rechte Seite auslöscht. Dadurch gilt für den relativen Fehler

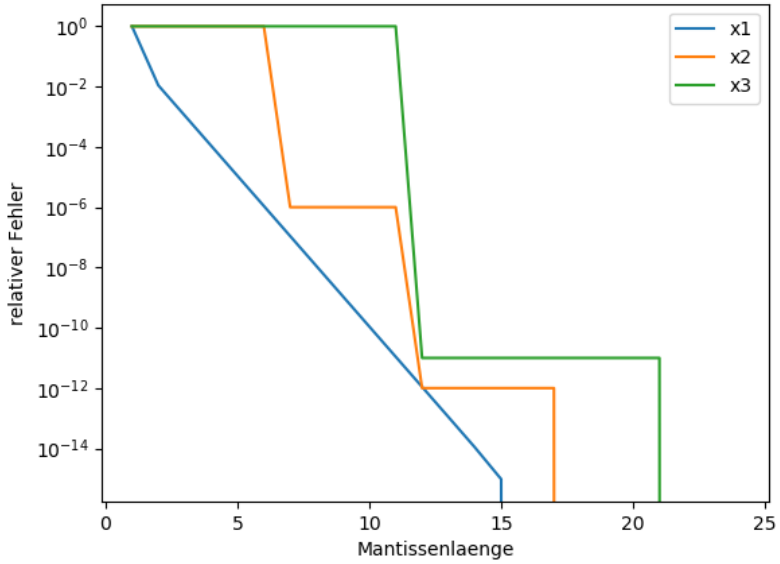
$$\epsilon(x) = |\frac{0 - yl}{yl}| = 1 \tag{4.1}$$

für den absoluten Fehler gilt bei Auslöschung der rechten Seite

$$\delta(x) = |0 - yl| = |yl| \tag{4.2}$$

Nun wollen wir und anschauen inwiefern die Fehler mit der Mantissenlänge zusammenhängen. Dazu betrachten wir x1 = 10, x2 = 10000000 und x3 = 100000000000. Den

Versuch werten wir grafisch anhand unseres mit `plotten()` ausgegebenen Diagramms aus.



Wir sehen, dass der relative Fehler streng monoton fallend in Abhängigkeit zur Mantissenlänge ist. Je kleiner der x-Wert ist, desto linearer verläuft der Graph. Außerdem wird der relative Fehler ab einer bestimmten Mantissenlänge $= 0$. Dies lässt sich wieder mit Auslöschungseffekten wie schon bei unserem ersten Experiment erklären.

4.2 Experimente zur relativen Maschinengenauigkeit

Für die drei verschiedenen Datentypen numpy.float16, numpy.float32 und numpy.float64 benutzen wir jeweils die Funktion *accurac1()* um die relative Maschinengenauigkeit zu ermitteln. Es ergeben sich die Werte:

```
numpy.float16:0.000977
numpy.float32: 1.1920929e-07
numpy.float64: 2.220446049250313e-16
```

Literaturverzeichnis

[1] Hella Rabus. *Einführung in das wissenschaftliche Rechnen.* Hella Rabus, 2019.