Computerphysik Programmiertutorial 4a

Prof. Dr. Matteo Rizzi und Dr. Markus Schmitt - Institut für Theoretische Physik, Universität zu Köln

Github: https://github.com/markusschmitt/compphys2022

Inhalt dieses Notebooks: Rechnen auf dem Rechner: Maschinengenauigkeit, Vergleichen von Fließkommazahlen

Rechnen auf dem Rechner: Maschinengenauigkeit

Zahlen werden im Computer in einem Binärcode dargestellt und für jede Zahl steht nur eine begrenzte Anzahl von Bits zur Verfügung. Es können daher weder alle ganzen Zahlen $\mathbb{Z}$ noch alle reellen Zahlen $\mathbb{R}$ dargestellt werden.

Ganze Zahlen - `Int`

Wir haben schon in einem früheren Tutorial gesehen, dass Ganzzahlen in 64 bits als Binärzahlen dargestellt werden. Das ergibt automatisch eine Grenze für die größte darstellbare Zahl. Schauen wir uns diese Grenze an:

In [1]:
```
zahl1 = 2^63
bitstring(zahl1)
```

Out[1]:
```
"1000000000000000000000000000000000000000000000000000000000000000"
```

In [2]:
```
zahl2 = 2^63-1
bitstring(zahl2)
```

Out[2]:
```
"0111111111111111111111111111111111111111111111111111111111111111"
```

In [3]:
```
zahl1
```

Out[3]:
```
-9223372036854775808
```

In [4]:
```
zahl2
```

Out[4]:
```
9223372036854775807
```

Die ganzen Zahlen auf dem Computer sind ein "Kreis":

In [5]:
```
start=2^63-4
for i in 1:6
    println("$start + $i = $(start+i)")
end
```

```
9223372036854775804 + 1 = 9223372036854775805
9223372036854775804 + 2 = 9223372036854775806
9223372036854775804 + 3 = 9223372036854775807
9223372036854775804 + 4 = -9223372036854775808
9223372036854775804 + 5 = -9223372036854775807
9223372036854775804 + 6 = -9223372036854775806
```

Fließkommazahlen - `Float`

Auf dem Computer können wir sehr leicht große Summen ausrechnen. Ein Beispiel ist die **Harmonische Reihe**

$$H_n = \sum_{k=1}^{n} \frac{1}{k}$$

Schreiben wir eine Funktion, die die n-te Harmonische Zahl H_n berechnet:

In [6]:
```
function H_forward(n, mytype=Float32)
    S = mytype(0.0)
    for k in 1:n
        S += mytype(1.0)/k
    end

    return S
end

println(H_forward(1000))
```

```
7.4854784
```

Da Addition kommutativ ist, können wir die Summe in beliebiger Reihenfolge ausrechnen, z.B. auch in umgekehrter Reihenfolge:

In [7]:
```
function H_backward(n, mytype=Float32)
    S = mytype(0.0)
    for k in n:-1:1
        S += mytype(1.0)/k
    end

    return S
end

println(H_backward(1000))
```

```
7.4854717
```

Mit unterschiedlicher Reihenfolge der Summation erhalten wir unterschiedliche Ergebnisse!

In [12]:
```
println(H_forward(100000)-H_backward(100000))
```

```
0.0006980896
```

Reelle Zahlen werden im Computer als **Fließkommazahlen** behandelt. Das bedeutet, dass sie bezüglich einer festen **Basis** b in **Vorzeichen** $\pm$, **Mantisse** m und **Exponent** e zerlegt werden. Eine reelle Zahl $r \in \mathbb{R}$ wird also geschrieben als

$$r = \pm m \times b^e$$

Das Vorzeichen wird in einem Bit kodiert, für Mantisse und Exponent steht jeweils eine feste Zahl weiterer Bits zur Verfügung. Das Kodieren der Mantisse in einer begrenzten Anzahl von Bits bedeutet, dass wir bei jeder Zahl nur eine feste Anzahl von **signifikanten Ziffern** kennen. Die begrenzte Anzahl von Bits für den Exponenten bedeutet, dass es wie bei Ganzzahlen auch eine größte und kleinste darstellbare Fließkommazahl gibt.

Da der Computer nur mit einer bestimmten Zahl von signifikanten Ziffern rechnet, ist der Unterschied zwischen Zahlen nur begrenzt auflösbar. Diese "Auflösung" können wir experimentell bestimmen, indem wir fragen was die kleinste Zahl ϵ ist, so dass auf dem Computer $1.0 + \epsilon > 1.0$:

In [13]:
```
eps = 1.0
n=0

while 1.0+eps > 1.0
    eps /= 2.0
    n += 1
end

println("n = $n")
println("eps = 1/2^$n = $eps")
println("1 + eps = $(1+eps)")
println("1 + 2eps = $(1+2eps)")
```

```
n = 53
eps = 1/2^53 = 1.1102230246251565e-16
1 + eps = 1.0
1 + 2eps = 1.0000000000000002
```

Die 64 bits des Datentyps `Float64` sind wie folgt aufgeteilt (Bild gestohlen von benjaminjurke.com):

Wir haben also 1 Bit für das Vorzeichen, 11 Bits kodieren den Exponenten als ganze Zahl zwischen -1022 und 1023. Als Basis wird $b = 2$ verwendet. Die darstellbaren Zahlen bewegen sich also (in etwa) zwischen $2^{-1022} \approx 10^{-308}$ und $2^{1023} \approx 10^{308}$. Die übrigen 52 Bits werden verwendet um die Mantisse als

$$m = 1 + \sum_{n=1}^{52} \mathrm{bit}_n \frac{1}{2^n}$$

zu kodieren.

Die folgende Funktion stellt eine gegebnene Fließkommazahl entsprechend dar.

In [14]:
```
using Printf

function maschinendarstellung(x::Float64)

    bits = bitstring(x)
    sgn = bits[1]
    exponent = bits[2:12]
    mantissa = bits[13:64]

    println("Dezimal              | Vorz.  Exponent     Mantisse")
    println(@sprintf("%.15e |   %s    %s  %s", x, sgn, exponent, mantissa))
    return nothing
end
```

Out[14]:
```
maschinendarstellung (generic function with 1 method)
```

Schauen wir uns also an wie $1 + \epsilon = 1$ zustande kommt:

In [15]:
```
maschinendarstellung(eps)
```

```
Dezimal              | Vorz.  Exponent     Mantisse
1.1102230246251565e-16 |   0    01111001010  0000000000000000000000000000000000000000000000000000
```

In [17]:
```
maschinendarstellung(-1.0)
```

```
Dezimal              | Vorz.  Exponent     Mantisse
-1.000000000000000e+00 |   1    01111111111  0000000000000000000000000000000000000000000000000000
```

In [20]:
```
maschinendarstellung(1.0+2eps)
```

```
Dezimal              | Vorz.  Exponent     Mantisse
1.000000000000000e+00 |   0    01111111111  0000000000000000000000000000000000000000000000000001
```

Beim Addieren zweier Zahlen unterschiedlicher Größenordnung geht Information über die kleinere Zahl verloren. Summationen sollten also immer so durchgeführt werden, dass nur ähnlich große Zahlen miteinander addiert werden.

Zurück zur Harmonischen Reihe. Welcher Summationsreihenfolge können wir also trauen? Für großes n gilt

$$H_n \approx \log(n) + \gamma + \frac{1}{2n} - \frac{1}{12n^2} + \frac{1}{120n^4}$$

mit der Euler-Gamma Konstante γ. Wir können also unsere beiden Ergebnisse damit vergleichen:

In [21]:
```
using Plots

function H_approx(n)
    n = Float64(n)
    return log(n)+Base.MathConstants.eulergamma+1.0/(2n)-1.0/(12n^2)+1.0/(120n^4)
end

n_werte = [2^n for n in 1:20]

Hn_fwd = [H_forward(2^n) for n in 1:20]
Hn_bwd = [H_backward(2^n) for n in 1:20]
Hn_approx = [H_approx(2^n) for n in 1:20]

plot(n_werte, abs.(Hn_fwd.-Hn_approx), label="forward", xaxis=:log)
plot!(n_werte, abs.(Hn_bwd.-Hn_approx), label="backward", xaxis=:log)
xlabel!("n")
ylabel!("Differenz")
```

Out[21]:

Vergleichen von Fließkommazahlen

Wegen der endlichen Maschinenpräzision müssen wir beim Vergleichen von Fließkommazahlen etwas vorsichtig sein. Dabei ergibt nämlich üblicherweise nur der *Vergleich innerhalb der numerischen Genauigkeit* Sinn.

Prüfen wir zum Beispiel naiv, ob 1.32 gleich $1.2 + 0.12$ ist:

In [22]:
```
1.32 == (1.2+0.12)
```

Out[22]:
```
false
```

In [23]:
```
1.32
```

Out[23]:
```
1.32
```

In [24]:
```
1.2+0.12
```

Out[24]:
```
1.3199999999999998
```

Zum Vergleich von Fließkommazahlen innerhalb der numerischen Genauigkeit gibt es die `isapprox()` Funktion:

In [25]:
```
isapprox(1.32, 1.2+0.12)
```

Out[25]:
```
true
```