Implementierung des Sinus Hyperbolicus

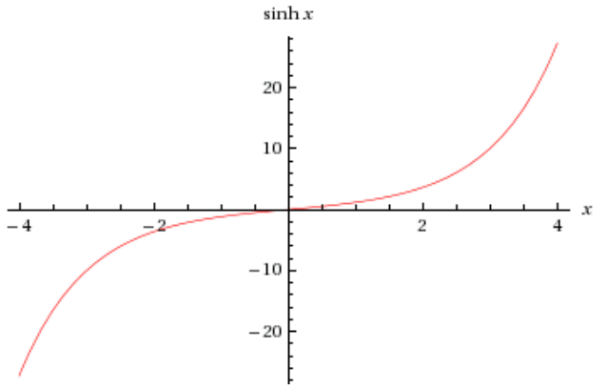
in x86-Assembly

Kevin Holm, Deniz Candas, Jakob Mezger

09.08.2017

1. Aufgabe
2. Verwendete Umgebung
3. Ansatz 1: Lookup Table
4. Ansatz 2: Reihenentwicklung
5. Code Review
    - Pseudo Code
    - Negative Eingabewerte
    - Anzahl Durchläufe der Hauptschleife

Aufgabe



- Implementierung des Sinus Hyperbolicus
- Erlaubte Befehle
  - x87 FPU-Befehle für Grundrechenarten, Negation
  - Speicherverwaltungsbefehle
- C-Rahmenprogramm
  - Validierung
  - Leistungsmessung
  - Vergleich mit Funktion der Standardbibliothek

Verwendete Umgebung

- Betriebssystem: Linux Ubuntu 64-Bit LTS 16.04
- Assembly-Syntax: nasm

Ansatz 1: Lookup Table

Und deren Vor- und Nachteile

- Stützstellen für Interpolation werden in Tabelle gespeichert
- Stützstellen können z.B. ganzzahlige Werte sein
- Werte zwischen Stützstellen lassen sich mit Polynom annähern

Ansatz 2: Reihenentwicklung

Und deren Vor- und Nachteile

- Reihenentwicklung: $\sinh(x) = \sum_{n=0}^{\infty} \frac{x^{2n+1}}{(2n+1)!}$
- Gesteigerte Effizienz durch Rekursion: $\sinh(x) = \sum_{n=0}^{\infty} \mathrm{term}_n$, $\mathrm{term}_0 = x,\ \mathrm{term}_{n+1}(x) = \mathrm{term}_n(x) \times \frac{x}{2n} \times \frac{x}{2n+1}$

Vorteile	**Nachteile**
Beliebige Genauigkeit (Interpolation nicht)	Viele Schleifendurchläufe für genaue Werte
Rekursion $\rightarrow$ Einfache Implementierung	Rechenintensiv
Wenige Speicherzugriffe	

Code Review

Pseudo Code

Sinh in Pseudocode

```
1   sinh(double x):
2   double i = 1
3   double result = x
4   double term = x
5
6   loop:
7   i++
8   term /= i
9   term *= x
10
11  i++
12  term /= i
13  term *= x
14
15  result = result + term;
16  if result is not precise enough jmp loop
17
18  return result
```

$$\sinh(x) = \sum_{n=0}^{\infty} \text{term}_n,\ \text{term}_0 = x,\ \text{term}_{n+1}(x) = \text{term}_n(x) \times \frac{x}{2n} \times \frac{x}{2n+1}$$

Code Review

Negative Eingabewerte

$\forall x \in \mathbb{R} : \sinh(-x) = -\sinh(x)$. Also kann man den sinh für den Betrag der Eingabe berechnen...

sinh.asm vor Aufruf der Hauptschleife

```
22  fld  st0                    ;st0 = x, st1 = x
23
24  fabs                        ;st0 = |x|
25
26  fdiv st1, st0               ;st1 = x/|x|, either 1 or -1
```

...und nach der Berechnung das Vorzeichen anpassen.

sinh.asm nach Verlassen der Hauptschleife

```
83  fincstp                     ;st6 = previous, st7 = i, st0 = result,
84                              ;st1 = 1, st2 = x, st3 = sign
85  fmul st0, st3               ;st0 = result * sign
```

Code Review

Anzahl Durchläufe der Hauptschleife

- Beobachtung: Größe der Eingabe und benötigte Schleifendurchläufe für genaues Resultat korrelieren
- Durch Ausprobieren: $45 + \frac{5|x|}{8}$ Durchläufe bieten guten Tradeoff

sinh.asm Bestimmung Anzahl Schleifendurchläufe

```
37   fld  st0                          ;I can only get a 64 Bit integer
38                                     ;out of the FPU with popping,
39                                     ;so duplicating x here to not loose it
40   fistp qword [rsp-8]               ;get x as integer into memory
41   fwait                             ;wait until that is done
42   mov rcx, qword [rsp-8]            ;then get that integer into rcx
43   shr rcx, 3                        ;divide it by 8
44   mov rax, rcx                      ;save that into rax
45   shl rcx, 2                        ;now multiply rcx with 4
46   add rcx, rax                      ;add rax to rcx
47   add rcx, 45                       ;add 45 to rcx, so now rcx = 45+5x/8
48   cmp rcx, 489                      ;compare rcx with 489
49   jbe .cont                         ;skip the next instruction if rcx <= 489
50   mov rcx, 489                      ;if rcx was bigger, set it to 489,
51                                     ;this is done so that inputs
52                                     ;> 710 don't run forever
53                                     ;and still output +/- infinity
```