Alle machen Fehler
auch Computer!

Walter Gander

ETH

gander@inf.ethz.ch

TecDay@KantiOlten

8. November, 2012

Zahlen

- Wieviele Zahlen sind im Intervall $[1, 10]$?

Zahlen

- Wieviele Zahlen sind im Intervall $[1, 10]$?
- Wieviele in $[1, 2]$

Zahlen

- Wieviele Zahlen sind im Intervall $[1, 10]$?
- Wieviele in $[1, 2]$?
- Wieviele in $[1, 1.00000000000000000000000000000000000000000000001]$?

Zahlen im Computer

- Wieviele Zahlen hat ein Computer im Intervall $[1, 2]$?

Zahlen im Computer

- Wie stellt ein Computer überhaupt Zahlen dar ?
- Wie speichert er sie ?

Relle Zahlen $\leftrightarrow$ Maschinenzahlen

- Mathematik: Relle Zahlen $\mathbb{R}$ = Kontinuum
  jedes Intervall $(a, b) \in \mathbb{R}$ mit $a < b$ enthält ∞-viele Zahlen

- Computer: Maschinenzahlen $\mathbb{M}$ = endliche Menge
  - Maschinenzahlen haben eine fest vorgegebene Anzahl Ziffern
  - ein Interval $(a, b) \in \mathbb{M}$ mit $a < b$ enthält nur endlich viele Maschinenzahlen

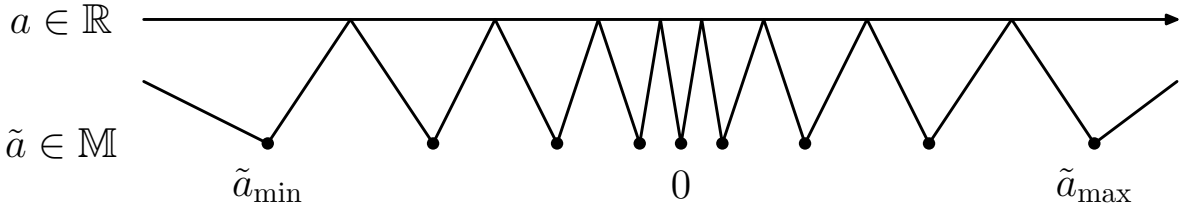
- unendlich $\Longleftrightarrow$ endlich

Maschinenzahlen

- alle reellen Zahlen mit den gleichen führenden Ziffern sind im Computer dieselbe Maschinenzahl
- z.B. bei Computer mit 6 Dezimalziffern:

$$\begin{array}{lcl} \mathbb{R} & & \mathbb{M} \\ 3.14159\,2653589793\ldots & & \\ 3.14159\,1122334455\ldots & \mapsto & 3.14159 \\ 3.14159\,0056748392\ldots & & \\ \ldots & & \end{array}$$

- Abbildung $\mathbb{R} \to \mathbb{M}$: ganzes Intervall $\in \mathbb{R} \mapsto \tilde{a} \in \mathbb{M}$:



Wie viele Maschinenzahlen in $[1, 2]$?

- Bei einem Computer mit nur 2 Dezimalstellen
  $1.0,\ 1.1,\ 1.2,\ 1.3,\ 1.4,\ 1.5,\ 1.6,\ 1.7,\ 1.8,\ 1.9,\ 2.0$
  10 Zahlen (ohne 2.0) im Abstand $1/10$

- Computer rechnen im Zweiersystem. Mit 5 dualen Nachkommastellen erhalten wir:

$Nummer$	$Dualzahl$		$Dezimal$
0	1.00000	$=$	1.0000
1	$1.00001 = 1 + 2^{-5}$	$=$	1.0312
2	$1.00010 = 1 + 2^{-4}$	$=$	1.0625
3	$1.00011 = 1 + 2^{-4} + 2^{-5}$	$=$	1.0938
	.....		
31	$1.11111 = 1 + 2^{-1} + 2^{-2} + \cdots + 2^{-5}$	$=$	1.9688

Somit 32 Zahlen (ohne 2) im Abstand von $2^{-5} = 1/32 = 0.0312$

Fliesskommadarstellung

- Zahl $= 8.881784197001252e{-}15$

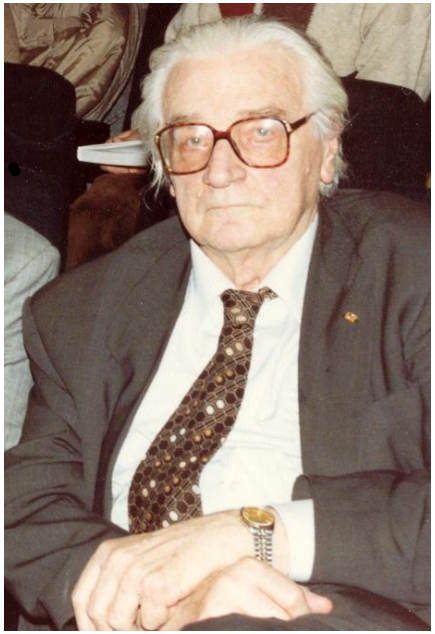
  Mantisse $= 8.881784197001252$

  Exponent $= -15$

  $= 8.881784197001252 \times 10^{-15}$

  $= 0\underbrace{.000000000000008}_{\leftarrow 15 \text{ Stellen}}881784197001252$

  Dezimalpunkt um Exponenten verschieben



Konrad Zuse
(1910–1995)
Computer Erfinder

- $8.881784197001252e{+}5 = 8.881784197001252 \times 10^{5}$

  $= 8\underbrace{88178}_{5 \rightarrow}.4197001252$

- Computer: Nicht Zehner- sondern Dualsystem.
  IEEE Floating Point Standard (seit 1985)

IEEE Floating Point Standard (seit 1985)

- Darstellung einer Maschinenzahl mit 64-Bit

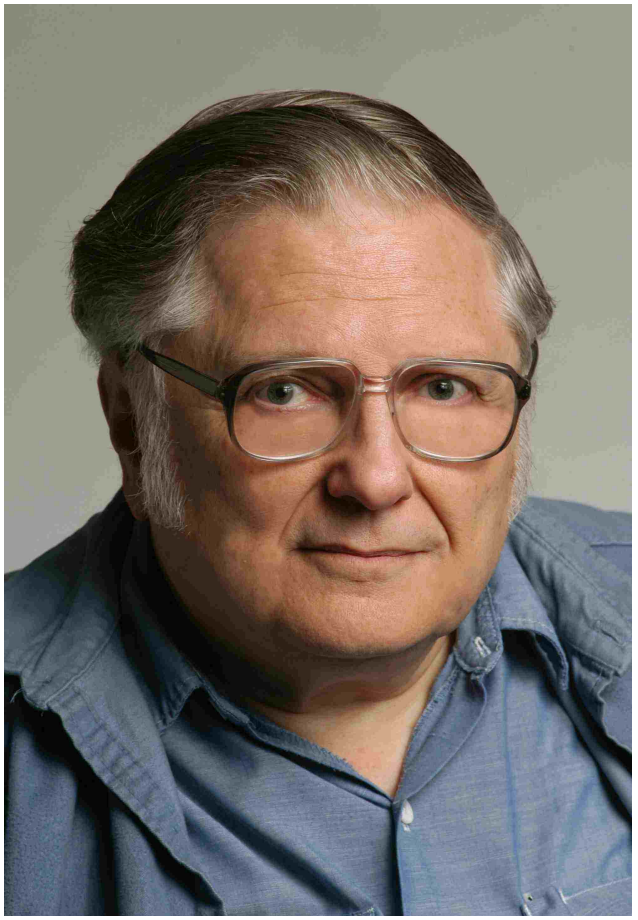
$$\begin{array}{cccc} & \overbrace{\qquad\qquad}^{e} & & \overbrace{\qquad\qquad\qquad}^{m} \\ S & EEEEEEEEEEE & & FFFFF\cdots FFFFF \\ 0 & 1 \qquad\qquad 11 & 12 & \qquad\qquad 63 \end{array}$$

S	1 Bit für das Vorzeichen
e	11 Bits für den Exponenten
m	52 Bits für die Nachkommastellen

- Normalfall: $0 < e < 2047, \quad (2^{11} = 2048)$

$$\tilde{a} = (-1)^S \times 2^{e-1023} \times 1.m$$

- Abstand Maschinenzahlen in $[1, 2]$:
  $eps = 2.2204e{-}16$ also
  1/eps = 4'503'599'627'370'495
  = 4.5 Billiarden Maschinenzahlen in $[1, 2)$



William Kahan (*1933)
Vater IEEE Floating Point System

Wegen endlicher Arithmetik, Rundung nach jeder Operation

$\Longrightarrow$ Computer rechnen prinzipiell ungenau!

Anweisungen			Resultate		
a	=	10	a	=	10
b	=	a/7	b	=	1.428571428571429
c	=	sqrt(sqrt(sqrt(sqrt(b))))	c	=	1.022542511383932
d	=	exp(16*log(c))	d	=	1.428571428571427
e	=	d*7	e	=	9.999999999999991
a-e			ans	=	8.881784197001252e-15

Studium/Kontrolle der Rundungsfehler $\Longrightarrow$ Numerische Mathematik

Richtiges Programm – trotzdem falsche Resultate!

Berechnung von π als Grenzwert der Flächen von regelmässigen n-Ecken

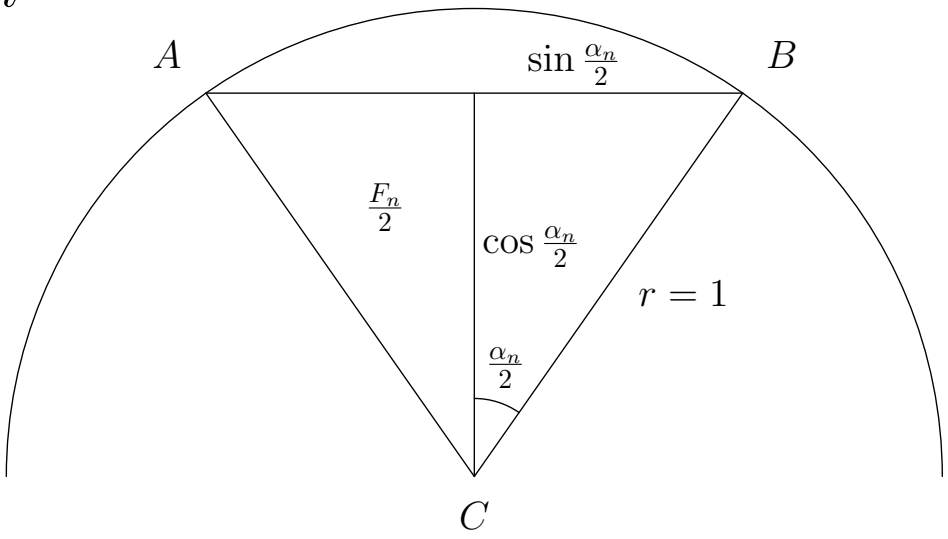
- A_n: Fläche n-Eck, F_n: Fläche ΔABC

$$A_n = n\,F_n = n\,\frac{\sin\alpha_n}{2}, \quad \alpha_n = \frac{2\pi}{n}$$

- $\lim_{n\to\infty} A_n = \pi$
- $A_6 = \frac{3}{2}\sqrt{3} = 2.5981$
  $A_{12} = 3$
- Rekursion $A_n \to A_{2n}$

$$\sin\left(\frac{\alpha}{2}\right) = \sqrt{\frac{1-\sqrt{1-\sin^2\alpha}}{2}}$$

Nur 2 Wurzeln und
rationale Operationen



Programm

pinaive

```
s=sqrt(3)/2; A=3*s; n=6;          % initialization
z=[A-pi n A s];                   % store the results
while s>1e-10                     % termination if s=sin(alpha) small
  s=sqrt((1-sqrt(1-s*s))/2);      % new sin(alpha/2) value
  n=2*n; A=n*s/2;                 % A = new polygon area
  z=[z; A-pi n A s];
end
m=length(z);
for i=1:m
  fprintf('%10d %20.15f %20.15f %20.15f\n',z(i,2),z(i,3),z(i,1),z(i,4))
end
```

Numerische Auslöschung

$$\begin{array}{rcl} 1.2345e0 & & \\ -1.2344e0 & & \\ \hline 0.0001e0 & = & 1.0000e{-}4 \end{array}$$

- Falls beide Zahlen exakt $\Longrightarrow$ Resultat $1.0000e{-}4$ exakt
- Falls beide Zahlen mit Rundungsfehler behaftet, dann

$$\begin{array}{rcl} 1.2345e0 & & \\ -1.2344e0 & & \\ \hline 0.0001e0 & = & 1.xxxxe{-}4 \quad \text{falsch!} \end{array}$$

Stabilisierung durch umformen Vermeidung von Auslöschung

$$\sin\frac{\alpha_n}{2} = \sqrt{\frac{1-\sqrt{1-\sin^2\alpha_n}}{2}} \quad \text{instabile Rekursion}$$

$$= \sqrt{\frac{1-\sqrt{1-\sin^2\alpha_n}}{2}\,\frac{1+\sqrt{1-\sin^2\alpha_n}}{1+\sqrt{1-\sin^2\alpha_n}}}$$

$$= \sqrt{\frac{1-(1-\sin^2\alpha_n)}{2(1+\sqrt{1-\sin^2\alpha_n})}}$$

$$\sin\frac{\alpha_n}{2} = \frac{\sin\alpha_n}{\sqrt{2(1+\sqrt{1-\sin^2\alpha_n})}} \quad \text{stabile Rekursion}$$

Programm

pistabil

```
oldA=0;s=sqrt(3)/2; newA=3*s; n=6;     % initialization
z=[newA-pi n newA s];                  % store the results
while newA>oldA                        % quit if area does not increase
   oldA=newA;
   s=s/sqrt(2*(1+sqrt((1+s)*(1-s)))); % new sin-value
   n=2*n; newA=n/2*s;
   z=[z; newA-pi n newA s];
end
m=length(z);
for i=1:m
  fprintf('%10d %20.15f %20.15f\n',z(i,2),z(i,3),z(i,1))
end
```

Elegantes Abbruchkriterium!

Quadratische Gleichungen

- Gegeben $x^2 + px + q = 0$, gesucht Lösungen x_1 und x_2
- Berechnung der Lösungen

$$x_{1,2} = -\frac{p}{2} \pm \sqrt{\left(\frac{p}{2}\right)^2 - q}.$$

- Faktorisierung $x^2 + px + q = (x - x_1)(x - x_2)$
- Naives Programm

```
function [x1,x2]=QuadGleichung(p,q)
diskriminante=(p/2)^2-q;
if diskriminante<0
    error('Loesungen sind komplex')
end
d=sqrt(diskriminante);
x1=-p/2+d; x2=-p/2-d;
```

Test von QuadGleichung

qnaive

- $(x-2)(x+3) = x^2 + x - 6 = 0$

  ```
  >> [x1,x2]=QuadGleichung(1,-6)

  x1=2, x2=-3
  ```
  richtig

- $(x-10^9)(x+2\cdot 10^{-9}) = x^2 + (2\cdot 10^{-9} - 10^9)x + 2$

  ```
  >> [x1,x2]=QuadGleichung(2e-9-1e9,2)

  x1=1.0000e+09, x2=0
  ```
  falsch

- $(x+10^{200})(x-1) = x^2 + (10^{200} - 1)x - 10^{200}$

  ```
  >> [x1,x2]=QuadGleichung(1e200-1,-1e200)

  x1=Inf, x2=-Inf
  ```
  falsch

Bessere Formel für Computer

$$x_{1,2} = -\frac{p}{2} \pm \sqrt{\left(\frac{p}{2}\right)^2 - q}.$$

Wenn $|p| > 1$ ausklammern (vermeiden Ueberlauf)

$$x_{1,2} = -\frac{p}{2} \pm |p|\sqrt{\frac{1}{4} - q/p/p}$$

Vermeiden Auslöschung durch Vieta ($|x_1| \geq |x_2|$)

$$x_1 = -\text{sign}(p)\left(|p|/2 + |p|\sqrt{\frac{1}{4} - q/p/p}\right)$$

$$x_2 = q/x_1 \qquad \text{Vieta}$$

Narrensicheres Programm für quadratische Gleichung qprofi

```
function [x1,x2]=quadeq(p,q)
if abs(p/2)>1                       % Vermeiden von Ueberlauf
  fak=abs(p); diskr=0.25-q/p/p; % durch Ausklammern
else
  fak=1; diskr=(p/2)^2-q;
end
if diskr< 0
  error('Loesungen sind komplex')
else
  x1=abs(p/2)+fak*sqrt(diskr);  % Berechnen absolut groessere Loesung
  if p>0, x1=-x1; end           % Vorzeichen anpassen
  if x1== 0, x(2)=0;
  else
     x2=q/x1;                   % Vermeiden Ausloeschung durch Vieta
  end                           % bei zweiter Loesung
end
```

Test von quadeq

- $(x-2)(x+3) = x^2 + x - 6 = 0$

  ```
  >> [x1,x2]=quadeq(1,-6)      x1 =  2     x2 = -3
  ```

  richtig

- $(x - 10^9)(x + 2 \cdot 10^{-9}) = x^2 + (2 \cdot 10^{-9} - 10^9)x + 2$

  ```
  >> [x1,x2]=quadeq(2e-9-1e9,2)   x1=1.0000e+09   x2=2.0000e-09
  ```

  richtig!

- $(x + 10^{200})(x - 1) = x^2 + (10^{200} - 1)x - 10^{200}$

  ```
  >> [x1,x2]=quadeq(1e200-1,-1e200)     x1=-1.0000e+200    x2=1
  ```

  richtig!

Lineare Gleichungen

$$ax + 3y = 6 \qquad (a \text{ ist eine kleine Zahl}) \tag{1}$$

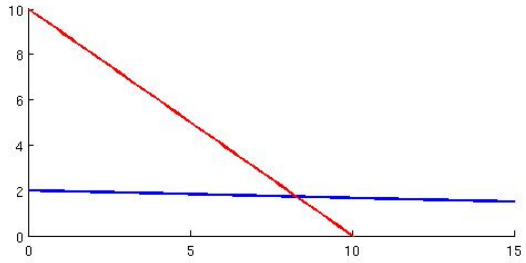
$$x + y = 10 \tag{2}$$

Lösung: eliminieren x aus (2). Multiplizieren (2) mit a und subtrahieren ergibt die neue zweite Gleichung:

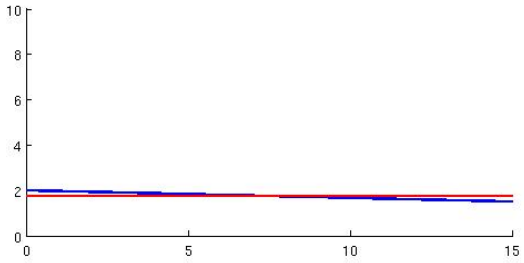
$$\begin{aligned} ax &+ 3y &&= 6 \\ &\quad (3-a)y &&= 6 - 10a \end{aligned}$$

Geometrische Interpretation: eine lineare Gleichung stellt eine Gerade dar

erstes System ($a = 0.1$)



transformiertes System: schleifender Schnitt!

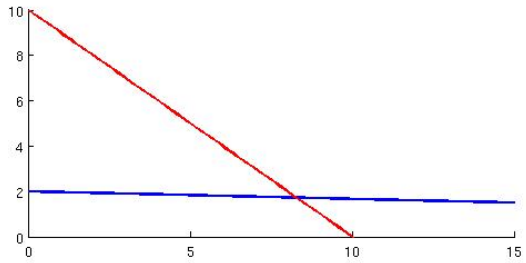


Unstabile Algorithmen vermeiden!

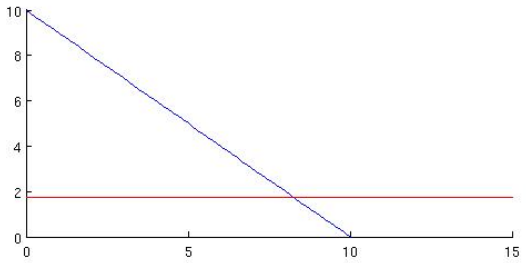
- gut konditioniertes Problem (sauberer Schnitt) nicht in schlecht konditioniertes (schleifender Schnitt) transformieren
- Pivotstrategie: Gleichungen vertauschen – Gleichung mit grössten Koeffizienten von x als erste verwenden:

$$\begin{aligned} x + y &= 10 \\ ax + 3y &= 6 \end{aligned} \quad \text{eliminieren} \Longrightarrow \quad \begin{aligned} x \quad + \quad y \quad &= \quad 10 \\ (a-3)y \quad &= \quad 10a - 6 \end{aligned}$$

erstes System ($a = 0.1$)



transformiertes System
auch gut konditioniert!



Zahlenbeispiel

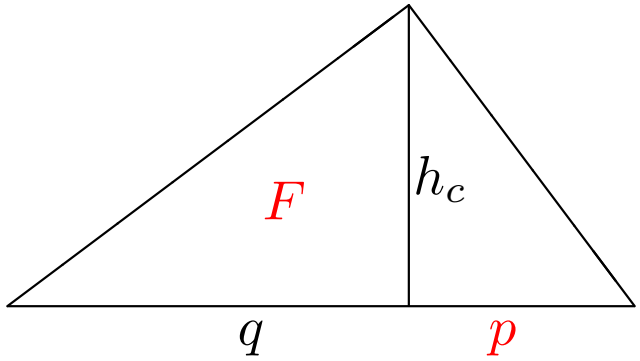
- $$\begin{aligned} ax + 3y &= 6 \\ x + y &= 10 \end{aligned}$$ mit kleinem $a = 1e - 12$
- Resultate

ohne Vertauschen:	mit Vertauschen
$x = 7.999823026239028$ falsch	$x = 8.000000000002666$
$y = 1.999999999997333$	$y = 1.999999999997333$

- ohne Vertauschen: falsches Resultat wegen der endlichen Arithmetik. Mathematisch sind beide Wege korrekt!

Nichtlineare Gleichungen

- Gegeben rechtwinkliges Δ mit $F = 12$, $p = 2$. Gesucht sind die Seiten.



- Lösung: $c = p + q$, Höhensatz: $h_c^2 = pq$, Ersetzen c und h_c in $F = \frac{c}{2}\, h_c$
- so folgt

$$F = \frac{p+q}{2}\sqrt{pq}$$

- Zahlen einsetzen, $\Longrightarrow$ Gleichung für $x = q$

$$f(x) = \frac{2+x}{2}\sqrt{2x} - 12 = 0$$

- $f(2) = -8, \quad f(8) = 8 \implies 2 < x < 8$

Bisektionsalgorithmus

- $f(a) < 0, \quad f(b) > 0 \implies a < x < b$
- probieren mit $x = (a + b)/2$
  wenn $f(x) < 0$ verschieben $a = x$ sonst $b = x$
- iterieren so lange bis $b - a < tol$
- ```
  function x=bisektnaive(f,a,b,tol)
  while b-a > tol
    x=(a+b)/2;
    if f(x) < 0, a=x; else b=x; end
  end
  ```
- Funktioniert nicht für hohe Genauigkeit??

`dreieck`
  ```

Bisektion narrensicher!

dreieck2

```
function x=Bisektion(f,a,b)
fa=f(a); v=1; if fa>0, v=-1; end;  % Bestimmung Verlauf von f
if fa*f(b)>0
  error('f(a) und f(b) haben dasselbe Vorzeichen')
end
x=(a+b)/2;
while (a < x) & (x<b)                % solange noch x in (a,b)
  if v*f(x)>0, b=x; else a=x; end; % weiter iterieren
  x=(a+b)/2
end
```

Entwicklung eines guten Algorithmus

- Beispiel: Berechnung von Quadratwurzeln

$$x = \sqrt{a} \iff x^2 = a,\ x > 0$$

  unter Verwendung der Grundoperationen $\{+, -, \times, /\}$

- Methode: Schätzen und korrigieren. Suchen x so dass

$$\frac{a}{x} = x$$

- Start mit Anfangswert x_1, berechnen $\frac{a}{x_1}$

$$\text{wenn} \quad \frac{a}{x_1} \neq x_1 \quad \text{nehmen Mittelwert} \quad x_2 = \frac{1}{2}\left(x_1 + \frac{a}{x_1}\right)$$

- iterieren und erhalten Folge $\{x_k\}$, die gegen $\sqrt{a}$ konvergiert

$\sqrt{20} = ?$

Approximation	dividieren		Mittelwert
4	$\frac{20}{4} = 5$	zu gross	$4.5 = \frac{4+5}{2}$
4.5	$\frac{20}{4.5} = 4.4444$	zu klein	$4.4722 = \frac{4.5+4.4444}{2}$
4.4722	$\frac{20}{4.4722} = 4.4721$	zu klein	$\ldots$

- Folge $x_k \to \sqrt{a}$ für $k \to \infty$

$$x_{k+1} = \frac{1}{2}\left(x_k + \frac{a}{x_k}\right)$$ Algorithmus von Heron

- Anfangswert? Abbruchkriterium?

Alternative Herleitung der Heron Iteration

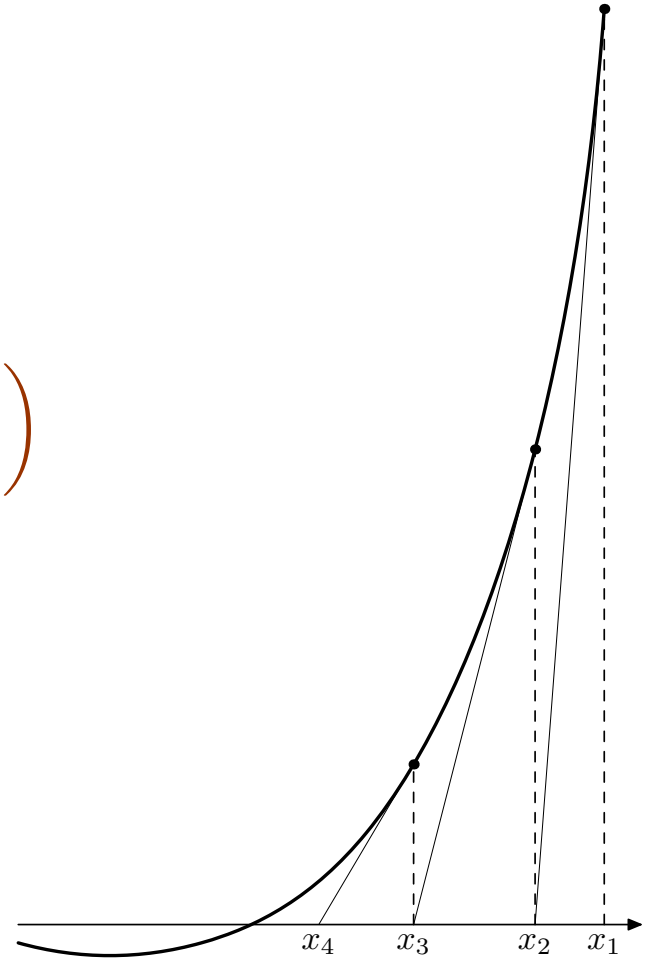
löse $f(x) = x^2 - a = 0$ mit der Newton Methode

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

$$f(x) = x^2 - a, \quad f'(x) = 2x$$

$$\Rightarrow x_{k+1} = x_k - \frac{x_k^2 - a}{2x_k} = \frac{1}{2}\left(x_k + \frac{a}{x_k}\right)$$

- Wenn $x_1 > \sqrt{a}$ dann monotone Konvergenz: $\sqrt{a} < \cdots < x_2 < x_1$
- Mit $x_0 > 0$ ist $x_1 > \sqrt{a} \implies \{x_k\}$ monoton fallend
- Startwert: $x_0 = 1$, $x_1 = (1 + a)/2$



Programm mysqrt mit schlauem Abbruchkriterium

Iteration abbrechen, wenn die monotone Konvergenz verletzt wird!

```
function xnew=mysqrt(a);
% computes w=sqrt(a) using Heron's algorithm
xold=(1+a)/2;              % start > sqrt(a)
xnew=(xold+a/xold)/2;      % first iterate
while xnew<xold            % if monotone
  xold=xnew;               % iterate
  xnew=(xold+a/xold)/2;
end
>> a= 12345.654321;
>> RelErr=(sqrt(a)-mysqrt(a))/sqrt(a)
RelErr = 1.2790e-16
```

Relativer Fehler kleiner als Maschinengenauigkeit $\varepsilon = 2.22 \cdot 10^{-16}$

Kleinste positive Maschinenzahl

- Kleinste normalisierte Zahl

```
>> realmin  =    2.225073858507201e-308
>> realmin =   0010000000000000  (im format hex)
```

- Division duch 2 ergibt nichtnormalisierte Zahl

```
>> x=realmin/2
x =    1.112536929253601e-308
>> x =   0008000000000000  (im format hex)
```

- Kleinste nichtnormalisierte Zahl (eps = 2.220446049250313e-16 Maschinengenauigkeit)

```
>> x= realmin*eps =  4.940656458412465e-324
x =   0000000000000001  (format hex)
>> x/2 = 0000000000000000
```

Kleinste normalisierte Maschinenzahl berechnen

Programm 1

```
function a=test1
a=1;
while a>0
  last=a; a=a/2.0;
end;
a=last;

>> test1
ans =  4.940656458412465e-324
```

falsch ergibt kleinste

nichtnormalisierte Zahl!

Programm 2

```
function a=test2
a=1;
while a*eps>0
  last=a; a=a/2.0;
end;
a=last;

>> test2
ans =  4.940656458412465e-324
```

immer noch falsch ???

haben doch mit a*eps nichtnormalisierte Zahlen übersprungen?!

Was ist los? Drucken von Zwischenresultaten

```
function a=test3
a=1;
while a*eps>0
  last=a; a=a/2.0;
  fprintf('%20.15e\n',a)  % Zwischenresultat drucken
end;
a=last;
```

```
>> a=test3
.....
4.450147717014403e-308
2.225073858507201e-308
1.112536929253601e-308
a = 2.225073858507201e-308
```

richtiges Resultat!!! Drucken verändert Resultate??? $\Longrightarrow$ Heisenbergeffekt

Zusammenfassung: was haben wir gelernt

- Mathematik: jedes Intervall enthält unendlich viele Zahlen,
  Computer: kann nur endlich viele Zahlen darstellen und endlich viele Operationen durchführen

  Philosophieren über Endlichkeit und Unendlichkeit
- Rundungsfehler: Computer rechnen prinzipiell ungenau
- Stabilität: mathematisch richtige Programme mit falschen Resultaten
- Kondition: ein gut konditioniertes Problem wird durch einen schlechten Algorithmus in ein schlecht konditioniertes transformiert
- Es ist spannend, für Computer richtig funktionierende Programme zu konstruieren. Eigenschaften der endlichen Arithmetik nützen!

. . . und die Moral von der Geschicht?

Dem Computer traue nicht!

neue Volksweisheit