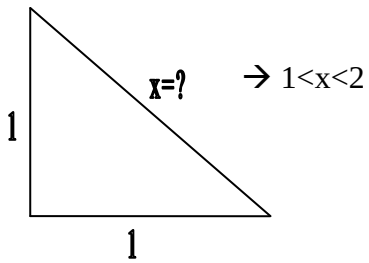
Unit 1
Eine Phantasiereise vom Babylon sqr(2) Algorithmus zum RSA Code

Die ca. 3800 Jahre alte Keilschrifttafel YBC (Yale Babylonian Collection) 7289, siehe „Bilder zu ybc 7289“ https://www.google.de/search?q=ybc+7289&tbm=isch&tbo=u&source=univ&sa=X&ei=ytNiUpSYFIjBtAbul4Eg&sqi=2&ved=0CDEQsAQ&biw=1400&bih=906&dpr=1
zeigt, dass die babylonischen Mathematiker in der Lage waren, die näherungsweise numerische Berechnung von $\text{sqr}(2) = \sqrt{2}$ mittels der 4 arithmetischen Grundoperationen A, S, M, D durchzuführen.
Sie erhielten sqr(2) ≈ 1.41421.
sqr(2) = 1.41421356237309504880168872420969807856967187 5…

Wie konnten die babylonischen Priester-Mathematiker ein solches Problem lösen? Im folgenden ein Rekonstruktionsversuch aus heutiger Sicht, gestützt auf [1],[2], [3], [4], [5] und darauf basierende „Variationen“.

Das babylonische sqr(2) Problem

Gegeben:
- die quadratische Gleichung $x^2 = 2$,
- = algebraische Beschreibung des geometrischen Problems



→ 1<x<2

Gesucht:
ein Verfahren für die (näherungsweise) Berechnung von

$x = \mathrm{sqr}(2) = \sqrt{2}$ in Dezimalschreibweise, nur unter Benutzung der 4 arithmetischen Grundoperationen A, S, M, D.
Wie haben die babylonischen Mathematiker dieser Problem gelöst?
Freier Rekonstruktionversuch:

Problemlösung Schritt für Schritt = Algorithmus Brückenschlag Mathematik - Informatik	Beispiel
• Geeignete Umformung von $x^2=2$ in "Variable = Ausdruck": $x = \frac{2}{x}$	
• These: wir wählen einen plausiblen Initialwert (Startwert) x[0]	x[0]=1
• Von der statischen Interpretation der Gleichung (Gleichheit, in beiden Richtungen) $x = \frac{2}{x}$ zu ihrer dynamischen Interpretation (gerichtete Wertzuweisung) $x[1] = \frac{2}{x[0]},\ x := \frac{2}{x}$ Der Wert des Ausdrucks $\frac{2}{x}$ for x=x[0] wird der linksstehenden Variablen x zugewiesen (als "neuer x-Wert" x[1]); • $x[1] = \frac{2}{x[0]}$ ist die Antithese to x[0]	x[1]=2

• These & Antithese werden zur Synthese zusammengefügt; hier als arithmetisches Mittel („vorantreibender Kompromiss“) $x[2]=\frac{1}{2}(x[0]+(x[1])=\frac{1}{2}(x[0]+\frac{2}{x[0]}).$ Mit anderen Worten: die ganze Vergangenheit dieser ersten Triade (These x[0] und Antithese x[1]) ist komprimiert in die Gegenwart, die Synthese x[2]; die Vergangenheit x[0] ist überschrieben mit x[2]. x[2] ist jetzt der Startwert für die Fortsetzung des Berechnungsprozesses, für die nächste Triade These & Antithese → Synthese $x[4]=\frac{1}{2}(x[2]+(x[3])=\frac{1}{2}(x[2]+\frac{2}{x[2]})$ und x[4] ist das Ergebnis der Kompression der Vergangenheit der zweiten Triade. x[2] wird überschrieben mit with x[4]. Und so fort.	$x[2]=\frac{3}{2}=1.5$ $x[4]=\frac{17}{12}=1.41\overline{6}$
• Die exhaustive Wiederholung der Triade These & Antithese → Synthese ist eine unendliche Schleife (infinite loop) mit dem Schleifenkörper = Rekursionsformel $x[2k+2]=\frac{1}{2}(x[2k]+\frac{2}{x[2k]});k=0,1,,2,...$ und dem Grenzwert $\lim_{k\to\infty} x[2k]=\mathrm{sqr}(2)=\sqrt{2}.$	

• Praktisch terminiert eine solche Schleife, wenn bei der Berechnung die Genauigkeit ε des realen Prozessors erreicht ist, wir haben eine bedingungsgesteuerte Schleife (condition controlled loop) mit einer datenabhängigen Haltbedingung (data depending termination condition; Pr3,1-15): erstmalig $\text{abs}(x[2k+2]-x[2k]) \le \varepsilon \rightarrow$ stop, k = n, Endwert abhängig von ε $\text{sqr}(2,\varepsilon) = x[2n]$.	Wertetabelle der Folge $< x[t] >_{t=0}^{10}$; x[0] = 1

Wertetabelle der Folge $< x[t] >_{t=0}^{10}$; x[0] = 1

t	x(t)
0	1
1	2
2	3/2 =1.5
3	4/3 =1.3per
4	17/12 =1.416per
5	24/17 =1.41176471…
6	577/408 =1.41421569…
7	816/577 =1.41421144…
8	665857/470832 = 1.41421356 …
9	941664/665857 = 1.41421356…
10	886731088897/ 627013566048 = 1.41421356 …

Mathematische Notation des babylonischen sqr(2) Algorithmus

$x[0]>0 \quad \| \ ? \ x[0]<0 \ ? \ \|$	Initialisierung
$x[2k+2]=\frac{1}{2}(x[2k]+\frac{2}{x[2k]});$ $k=0,1,2,...$	Rekursionsformel
Erstmalig $abs(x[2k+2]-x[2k]) \le \varepsilon$ $\rightarrow$stop, k = n	Haltbedingung
sqr(2,ε) = x[2n]	Ergebnis

Informatik-Notation (indexfrei), repeat until loop

Input: x[0]>0; ε>0	
x:=x[0]	
sequence loop Repeat	y : = x
	$x := \frac{1}{2}(y+\frac{2}{y})$
until Haltbedingung $\|x-y\| \le \varepsilon$ erstmalig erfüllt ist.	
sqr(2,ε):=x	
Output: sqr(2,ε)	

Experiment mit einer “verrückten” Initialisierung x[0] = 100 :

t	x[t]		Differenzenquotient $dx[2k]=\frac{x[2k]-x[2k+2]}{x[2k+2]-x[2k+4]}$
0	100,0		2.0080016

1	0,02		
2	50,01		
3	0,039992002		
4	25,024996		
5	0,079920093		
6	12,55245805		2.051425714
7	0,159331343		
8	6,355894695		2.208347451
9	0,314668524		
10	3,33528161		2.876798477
11	0,599649515		
12	1,967465563		6.275969469
13	1,016536217		
14	1,49200089		37.38779303
15	1,340481774		
16	1,416241332		$1395.535813 \approx 37.38779303^2$
17	1,412188696		
18	1,414215014		
19	1,414212111		
20	1,414213562		
21	1,414213562		
22	1,414213562 =sqr(2,ε=10^{-9})		

- Bsqr(2)A konvergiert für alle x[0]>0 to gegen den „Fixpunkt“ sqr(2), er ist korrekt und robust.
- Die Anzahl der Rekursionsschritte für die ε-Termination hängt von x[0] ab, ist kleiner für plausible, größer für implausible x[0].
- Bsqr(2)A konvergiert in Fixpunktferne linear, in Fixpunktnähe quadratisch, ist ein mathematischer „Selbstläufer“.

1. Erweiterung:

Vom Babylonischen sqr(2) Algorithmus zum Heron (ca. 60 AD) sqr(a) Algorithmus

Erweiterung: $x^2 = 2 \rightarrow x^2 = a \in \mathbb{R}_+$
Beibehaltung aller "guten" Eigenschaften: korrekt, robust, quadratische Konvergenz!

Mathematische Notation des Heron sqr(a) Algorithmus

$a, x[0] \in \mathbb{R}_+$

$$x[i+1] = \frac{1}{2}\left(x[i] + \frac{a}{x[i]}\right) ; i = 1,2,3,...$$

$$\lim_{i\to\infty} x[i] = sqr(a)$$

Haltbedingung: erstmalig $abs(x[i+1]-x[i]) \leq \varepsilon$

Computer science notation (index free), structogram, repeat until loop

Input: a; x[0]; ε	
x: = x[0]	
	y : = x
	$x:=\frac{1}{2}\left(y+\frac{a}{y}\right)$
$\lvert x - y\rvert \leq \varepsilon$	
sqr(a, ε): = x	
Output: sqr(a, ε)	

Frage: Wie reagiert der Heron Algorithmus für

a	x[0]
pos.	pos.
pos.	neg.
neg.	pos.
neg.	neg. ???

$$a(t) = a_1 + a_2(-1)^t$$

$$a(t) = a_1 + a_2(-1)^{f(t)}$$

Heron Algorithmus, Experiment: Sprung im "Input a"

i	"a[i]"	x[i]
0	$a_1 = 2$	100.0

1	2	50.01
2	2	25.024996
3	2	12.55245805
4	2	6.355894695
5	2	3.33528161
6	2	1.967465563
7	2	1.49200089
8	2	1.416241332
9	a_2 = 87	1.414215014
10	87	31.46622092
11	87	17.1155453
12	87	11.09932182
13	87	9.468819279
14	87	9.328435433
15	87	9.327379113
16	87	9.327379053 = sqr(87,ε= 10^{-9})
17	87	9.327379053

x-robust, a-robust, dynamischer Inputfolger, einfachstes Modell für einen Selbstläufer[1)] [6].
Wie kann ein Selbstläufer gestartet, gesteuert, verstärkt, gebremst, „umgelenkt“, „außer Tritt“, zur Oszillation gebracht, … werden?
Experiment: ?

2. Erweiterung:
Heron→Newton (1643 – 1727)
Newton’s zero–finding algorithm

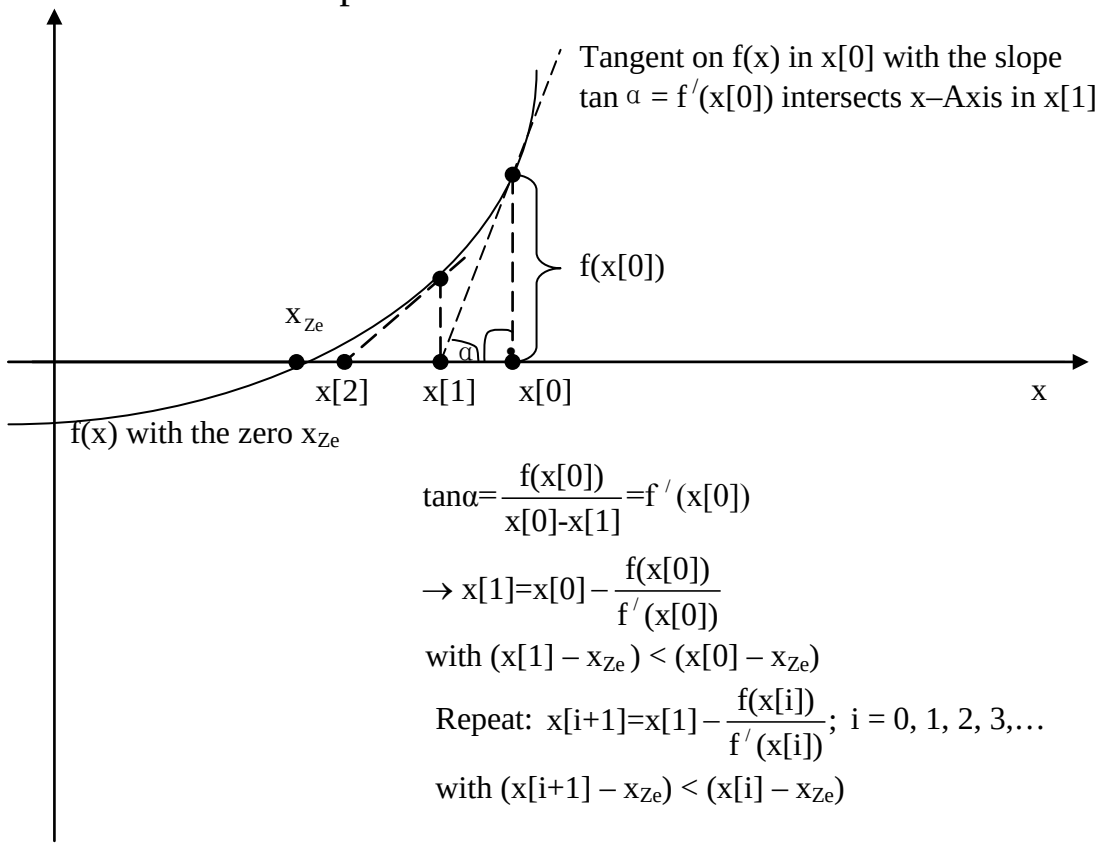
Erweiterungsidee:
• Geeignete Umformung der Heron Rekursionsformel

$$\underline{x[i+1]} = \frac{1}{2}\left(x[i] + \frac{a}{x[i]} \right) \Rightarrow x[i] - \frac{(x[i])^2 - a}{2x[i]} = x[i] - \underline{\frac{f(x[i])}{f'(x[i])}} \quad ; \quad i=0,\ 1,\ 2,\ldots$$

Hierbei ist
$f(x) = x^2 - a$ Funktion von x mit den Nullstellen $x_{1;2} = \pm\sqrt{a}$
= Wurzeln von $f(x) = x^2 - a = 0$
$f'(x) = 2x$ erste Ableitung von f(x)
• Newton: Erweiterung auf beliebige reelle Funktionen f(x)

$$x[i+1]=x[i]-\frac{f(x[i])}{f'(x[i])};\ i = 0, 1, 2,\ldots$$

- $\lim_{i\to\infty} x[i]=x_{Nu}$?? in allen Fällen von f(x) ??
- Geometrische Interpretation:



$$\tan\alpha=\frac{f(x[0])}{x[0]-x[1]}=f'(x[0])$$

$$\rightarrow x[1]=x[0]-\frac{f(x[0])}{f'(x[0])}$$

with $(x[1]-x_{Ze}) < (x[0]-x_{Ze})$

Repeat: $x[i+1]=x[1]-\frac{f(x[i])}{f'(x[i])};\ i = 0, 1, 2, 3,\ldots$

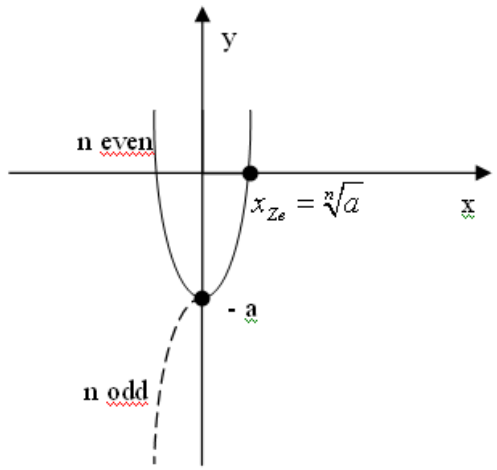
with $(x[i+1]-x_{Ze}) < (x[i]-x_{Ze})$

Discuss the convergence $\lim_{i\to\infty} x[i]=x_{Ze}$

depending on f(x) and x[0]!

Newton Spezialfall

$f(x) = x^n - a;\ n \in \mathbb{N};\ a, x[0] \in \mathbb{R}_+;\ x_{Nu}=\sqrt[n]{a}$



$$x[i+1]=x[i]-\frac{(x[i])^n-a}{n(x[i])^{n-1}}=x[i]*\frac{(n-1)(x[i])^n+a}{n(x[i])^n}\ ;\ i=0,1,2,\ldots$$

$$=x[i]*\frac{u[i]}{v[i]};\quad \begin{pmatrix} u[i] \\ v[i] \end{pmatrix}=\begin{pmatrix} n-1 & 1 \\ n & 0 \end{pmatrix}\begin{pmatrix} (x[i])^n \\ a \end{pmatrix}$$

$$\lim_{i\to\infty} x[i]=\sqrt[n]{a}$$

Struktogramm Notation:

Input: a; x[0]; ε	
x: = x[0]	
	y : = x
	$\begin{pmatrix} u \\ v \end{pmatrix}:=\begin{pmatrix} n-1 & 1 \\ n & 0 \end{pmatrix}\begin{pmatrix} y^n \\ a \end{pmatrix}$
	$x:=y*\frac{u}{v}$
$\lvert x-y\rvert \le \varepsilon$	
n^{te} Wurzel(a, ε): = x	
Output: n^{te} Wurzel(a, ε)	

Erweiterung: Symmetrisierung der Kopplungsmatrix

$$\frac{1}{2}\left(\begin{pmatrix} n-1 & 1 \\ n & 0 \end{pmatrix}+\begin{pmatrix} 0 & n \\ 1 & n-1 \end{pmatrix}\right)=\frac{1}{2}\begin{pmatrix} n-1 & n+1 \\ n+1 & n-1 \end{pmatrix},$$

Objekt Spiegelbild

These & Antithese → Synthese
Tuning eines Algorithmus → Konvergenzbeschleunigung

Konvergenztest
n=6; a=5; x[0]=7; $\sqrt[6]{5} = 1.307660486...$

Ohne Tuning

i	x[i]	dx[i]
0	7.0	
1	5.833382917	
2	4.861647689	
3	4.051679905	
4	3.377163129	
5	2.816199568	
6	2.351537346	
7	1.971203813	
8	1.670670042	
9	1.456252356	3.707146985
10	1.340787113	15.79044816
11	1.309640454	264.1269416
12	1.307667954	↓
13	1.307660486	in Fixpunktnähe:
14	1.307660486	$dx[i+1] \approx (dx[i])^2$ = quadratische Konvergenz

mit Tuning

i	x[i]	dx[i]
0	7.0	
1	5.000145707	
2	3.572316029	
3	2.555856636	
4	1.847782297	3.709367931
5	1.424175239	49.31999122
6	1.309975983	110260.7619
7	1.307660507	↓
8	1.307660486	in Fixpunktnähe:
9	1.307660486	$dx[i+1] \approx (dx[i])^3$ = kubische Konvergenz

?? Wie noch schneller? ?
tune = make small changes to an engine so that it works better

Experiment mit dem tuned Newton, Spezialfall
$f(x)=x^n - a;\ a=b^n;\ b,x[0] \in \mathbb{R}_+$

Input: b; x[0]; ε	
x : = x[0]	
	y : = x
	$\begin{pmatrix} u \\ v \end{pmatrix} := \begin{pmatrix} n-1 & n+1 \\ n+1 & n-1 \end{pmatrix} \begin{pmatrix} y^n \\ b^n \end{pmatrix}$
	$x := y * \frac{u}{v}$
$\lvert x - y \rvert \leq \varepsilon$	
? (b, ε) : = x	
Output : ? (b, ε)	

Experiment n=2; b=5; x[0]=50

i	x[i]
0	50.0
1	17.10963455
2	6.966075316
3	5.044552888
4	5.000000871
5	5.000000001
6	5.0

After a short transient process (shape and duration depend on b, x[0], and ε) we obtain ? (b,ε) = b, i.e. output = input → **dynamic input follower with cubic convergence**, strong attractor.

Application:

Algorithmic model

engine = fast converging dynamic input follower

+ load = ** slow converging sequential numerical algorithm
** solving numeral equations
** decomposition (splitting) of a numerical value b regarding a given expression containing k numerical variables (simplest case: expression = v1 +v2) into k values of this variables (simplest case: va1 and va2) so that the value of the given expression is b for this k values of the variables (simplest case: va1+va2 = b)
** factoring natural numbers with constraints

Experiment

Engine

Input: b; x[0]; ε	
x : = x[0]	
	y : = x
	$\begin{pmatrix} u \\ v \end{pmatrix} := \begin{pmatrix} n-1 & n+1 \\ n+1 & n-1 \end{pmatrix} \begin{pmatrix} y^n \\ b^n \end{pmatrix}$
	$x := y * \frac{u}{v}$
$\lvert x - y \rvert \leq \varepsilon$	
? (b, ε) : = x	
Output : ? (b, ε)	

Load = controllable decomposition of a positive number.
Experimental test:
n=2; b= 60; expression=$v_1 + v_2$;
dynamic input follower recursion formula:

$$x[i+1]=x[i-1]\frac{1+3\left(\frac{b}{x[i-1]+x[i]}\right)^2}{3+\left(\frac{b}{x[i-1]+x[i]}\right)^2};\ i=1,2,3,...$$

with the initial values
x[0] = x[1] =70 .

Record of the series $<x[i]>_{i=0}$

i	x[i]	x[i-1] + x[i]	b-comments
0	70.0		60.0
1	70.0	140.0	60.0
2	34.1025641	104.1025641	After a transient
3	41.94208978	76.04465388	process, b = 60 is
4	21.23010586	63.17219564	decomposed into
5	37.83559981	59.06570567	two summands,
6	21.90704831	59.74264812	21.86.. and 38.14..,
7	38.16226875	60.06931706	its sum is 60.
8	21.85651817	60.01878692	Decomposition

9	38.13838161	59.99489978	depends on b, x[0], x[1] ; can be changed by ← injection into x[i]
10	21.86023441	59.99861602	
11	38.14014112	60.00037553	
	→ +4 =		
	42.14014112	64.00037553	
12	20.49522576	62.63536688	
13	40.36790677	60.86313253	
14	20.20458676	60.57249353	
15	39.98638298	60.19096974	
16	20.1404832	60.12686618	
17	39.90201282	60.04249602	60.0
18	20.12622846	60.02824128	60.0
19	39.88324026	60.00946872	65.0 ← change of b
20	21.79720008	61.68044034	65.0
21	42.02819095	63.82539103	New transient process
22	22.19831065	64.2265016	
23	42.5343304	64.73264105	65.0
24	22.2899938	64.8243242	Decomposition of b into summands 22.31.. and 42.68..
25	42.64959946	64.93959326	
26	22.31072793	64.96032739	
27	42.67564647	64.9863744	
28	22.3154058	64.99105227	
29	42.68152188	64.99692768	
30	22.31646063	64.99798251	
31	42.68284647	64.9993073	65.0
		x[i-1]*x[i]	← Change of expression
		952.5300673	65.0
32	7.531049432	321.4466282	Fast factorization of b, cubic convergence!
33	15.75810547	118.6750712	
34	4.336007006	68.32725572	
35	14.99121665	65.00202042	
36	4.335872234	65.00000003	

How about integer factorization of $b \in \mathbb{N}$, esp. semiprime (= product of two primes)? → security guarantee of the RSA public key based cryptosystems [7].

References

[1] D. Fowler, E. Robson. Square Root Approximations in Old Babylonian Mathematics: YBC 7289 in Context. Historia Mathematica **25** (1998), 366-378
[2] H.L. Resnikoff, R.O. Wells. Mathematik im Wandel der Kulturen. Friedr. Vieweg & Sohn. Braunschweig/Wiesbaden 1983
[3] http://it.stlawu.edu/~dmelvill/mesomath/tablets/YBC7289.html
[4] http://en.wikipedia.org/wiki/Square_root_of_2
[5]https://www.google.de/search?q=ybc+7289&tbm=isch&tbo=u&source=univ&sa=X&ei=ytNiUpSYFIjBtAbul4Eg&sqi=2&ved=0CDEQsAQ&biw=1400&bih=906&dpr=1
[6] http://www.duden.de/rechtschreibung/Selbstlaeufer
[7] http://en.wikipedia.org/wiki/RSA_%28algorithm%29

1) A possible starting position for an integrative system theory of self-running phenomena
** Positive and negative feedback, recursion, chain reaction
** Self-running phenomena live cycle, mathematical models
** How we can trigger, start, influence, control, streamline, stop a self-running phenomenon?

Selbstläufer (Duden [6]): “Etwas, was wie von selbst, ohne dass viel dafür getan werden müsste, den gewünschten Erfolg hat.“

Self-running phenomena:
life
cancerogenese
pandemic
triumph march of the world religions
stampede, mass panic
rumor
neighbor’s disputes
marital disputes
corruption
hurricane, tornado
avalanche
dust explosion (coal, flour)
computer and ±application
internet
ebay, facebook, twitter, google, wiki,
fast-selling item

self-fulfilling prophecy
flow