Wilfried Rohm **wrohm@aon.at**

Programmieren in Mathcad (Eine Einführung)

- Mathematische / Fachliche Inhalte in Stichworten:

  Programmierung, Periodische Funktionen, Rekursion

- Kurzzusammenfassung

  An einfachen Beispielen (in 8 Schritten) werden der Umgang mit der Programmierpalette sowie Nutzen und Anwendung von Mathcad-Programmen erläutert. Alle Programme sind ausführlich kommentiert. Auf mögliche Fehlerquellen (insbesondere bei der Eingabe!) wird besonders hingewiesen!

- Lehrplanbezug (bzw. Gegenstand / Abteilung / Jahrgang):

  Angewandte Mathematik, alle Abteilungen; Schnittstelle zur EDV

- Mathcad-Version:

  Mathcad 2001

- Literaturangaben:

  Markus Hörhager, Heinz Partoll: Mathcad (Einführung, Anwendung, Referenz); Version 7, Addison-Wesley, 1998 (ISBN 3-8273-1343-0)

  Mathcad 2001, Das offizielle Benutzerhandbuch, Verlag mitp (ISBN 3-8266-0730-9)

  Quicksheets zum Thema "Programmieren" im Mathcad-Informationszentrum

- Anmerkungen bzw. Sonstiges:

  "Programmieren" in Mathcad muß man schon dann, wenn man nur stückweise definierte Funktionen darstellen will. Daher benötigt man elementare Kenntnisse auf diesem Gebiet recht häufig. Dazu muß man nicht EDV-mäßig vorgebildet sein.

  Schwierigere Programme (etwa für komplexe Algorithmen) dienen der allgemeineren und eleganten Formulierung von Problemen und sind sicherlich dem in Informatik vorgebildeten Mathcad-User vorbehalten. Der "einfache" Anwender von Mathcad wird dies kaum bzw. selten benötigen!

INHALT : (gewünschten Bereich anklicken!)

Grundsätzliches zu Mathcad-Programmen

zurück zum Inhalt

Nach der Auswahl der Symbolleiste "Programmierung" erhält man die nebenstehende Programmierungspalette, welche sämtliche Programmieroperatoren enthält.

Ein Mathcad-Programm wird zeilenweise in "**Programmzeilen**" editiert. Diese Programmzeilen können in beliebiger Anzahl mittels des Befehls **"Programmzeile hinzufügen" (="+1 Zeile")** erzeugt werden.

Eigentlich ist jedes "Programm" nichts anderes als eine **"Funktion"**, die einen Wert (Skalar, Vektor, Feld, Matrix, Zeichenfolge) zurückgibt. Als Wert der Funktion wird stets das Ergebnis der letzten Programmzeile angesehen, es sei denn, dass die Funktion durch den **return**-Operator bereits vorher verlassen wurde. Der Wert der Funktion kann mit "=" abgefragt werden. Grundsätzlich ist auch die symbolische Berechnung mittels des → Operators möglich, aber natürlich nicht immer durchführbar!

In Programmen definierte Variablen / Konstanten bzw. "Zwischengrößen" sind nur lokal gültig, d.h. außerhalb des Programmes unbekannt. Gegebenenfalls müssen daher für Größen, die global bekannt sein sollen, eigene Funktionen definiert werden. Innerhalb der Programme können alls vordefinierten Mathcad-Funktionen verwendet werden.

Zur Steuerung des Programmablaufes dienen die Verzweigungsoperatoren **if / otherwise** sowie die Schleifenbefehle **for** und **while**. **continue** und **break** sind Befehle, welche das (vorzeitige) Verlassen einer Schleife (break) bzw. die (frühzeitige) Fortsetzung der Schleife (continue) erlauben.

Die Fehlerbehandlung kann über den Befehl **on error** erfolgen, benutzerdefinierte Fehlermeldungen erlaubt die **Fehler(...)**-Funktion.

Über den Menüpunkt **Einfügen/Verweis...** können Programmsammlungen aus anderen Files übernommen werden!

Wichtige Hinweise zur Eingabe (Editieren von Mathcad-Programmen)

zurück zum Inhalt

ACHTUNG: Zu Beachtung bei der Eingabe von "Programmen":

1) Schlüsselwörter NIE selbst schreiben, immer aus der Symbolpalette holen (oder Kürzel verwenden)

2) Sind mehrere symbolische Wörter in einer Zeile, ist die Reihenfolge der Eingabe wichtig! Grundsätzlich wird in jener Reihenfolge eingegeben, in welcher der Computer den Ausdruck verarbeitet; dies ist insbesondere bei if / otherwise - Konstrukten wichtig, da zuerst der if-Ausdruck ausgewertet wird, und dann entschieden wird, ob die links stehende Anweisung ausgeführt wird.

DAHER:
ZUERST "if" drucken, DANN "return" / "break" etc., DANN die Platzhalter ausfüllen!!!

3) Die benutzerdefinierte Fehlerfunktion wird Fehler("...") geschrieben und **nicht** fehler("...") oder fehl("..."), wie in der Hilfe bzw. im Handbuch fälschlicherweise geschrieben steht!!

Schritt 1: Verwendung nur lokal gültiger Definitionen / Konstanten

zurück zum Inhalt

$Z(R, L, f) := R + j \cdot 2 \cdot \pi \cdot f \cdot L$

$Z\left(\frac{R}{10}, L, f_2\right) \rightarrow \frac{1}{10} \cdot R + 2 \cdot i \cdot \pi \cdot f_2 \cdot L$

$R \leftarrow 200\Omega \quad = 200 + 3.142i\,\Omega$
$L \leftarrow 1mH$
$f \leftarrow 500Hz$
return $Z(R, L, f)$

$Z(R, L, f) \rightarrow R + 2 \cdot i \cdot \pi \cdot f \cdot L$

In diesem Beispiel wird eine Funktion in Abhängigkeit verschiedenener Größen definiert. Es sei angenommen, dass diese Funktion in weiterer Folge für verschiedene **symbolische** Auswertungen verwendet werden soll.

Zwischendurch soll hier eine numerische Auswertung mit speziellen Werten für R und L erfolgen.
Werden diese Werte in "Programmzeilen" definiert, so gelten sie nur **lokal** innerhalb des jeweiligen "Programmes"!

Das "return" ist hier eigentlich überflüssig, erhöht aber meiner Ansicht nach die Übersicht!

Außerhalb des Programmes kann wieder weiter allgemein mit symbolischen Größen bzw. den ursprünglichen Werten gerechnet werde!

Schritt 2: Definition stückweise definierter Funktion

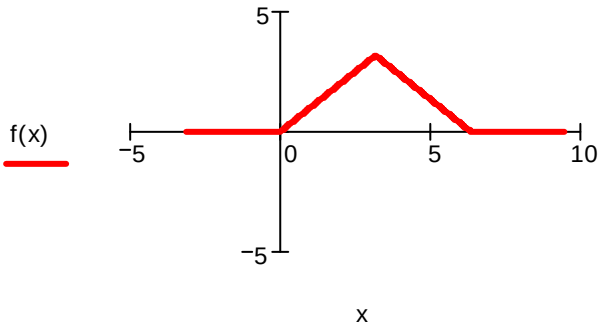
zurück zum Inhalt

$f(x) :=$
$x \quad \text{if} \quad 0 \le x < \pi$
$2\pi - x \quad \text{if} \quad \pi \le x \le 2 \cdot \pi$
$0 \quad \text{otherwise}$

Man achte darauf, dass wirklich alle Werte des in Frage kommenden Bereiches definiert sind!

In jeder Zeile könnte (aber muß nicht) hier am Anfang der "return"-Operator stehen.

$x := -\pi, -\pi + 0.01 .. 3 \cdot \pi$



Schritt 3: Definition und Zeichnen einer periodischen Funktion

zurück zum Inhalt

$T := 2 \cdot \pi$

$f(x) := x - \pi$

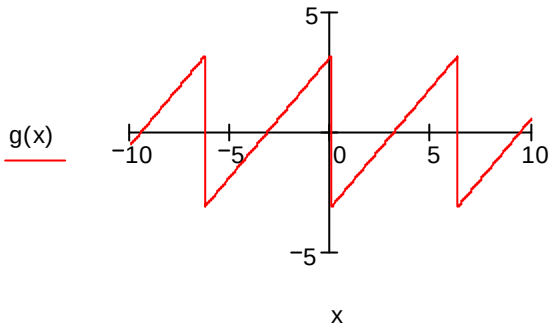
$g(x) := f\left(x - T \cdot \text{floor}\left(\frac{x}{T}\right)\right)$

$x := -10, -9.99 .. 10$

Festlegung der Periodenlänge für die folgenden Beispiele

Diese **Sägezahnschwingung** läßt sich als "periodische Fortsetzung" der Grundfunktion f(x) definieren.

die floor(z)-Funktion liefert die gegenüber z nächstkleinere ganze Zahl. Damit wird gezählt, wie oft die Periode 2*π abgezogen (für x>0) bzw. dazugezählt (für x<0) werden muß, um in den Bereich [0 - 2π] zu gelangen

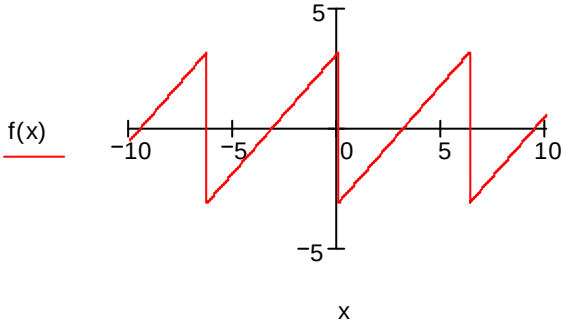


Diese Sägezahnschwingung hätten wir aber auch über ein "Programm" definieren können, beispielsweise so:

$$f(x) := \begin{cases} x - \pi & \text{if } 0 \le x < T \\ f\left(x - T \cdot \text{floor}\left(\frac{x}{T}\right)\right) & \text{otherwise} \end{cases}$$

Hier wird ein **rekursiver Funktionsaufruf** verwendet!!

Im Gegensatz zur obigen Methode ist eine symbolische Berechnung der Funktion hier NICHT möglich!

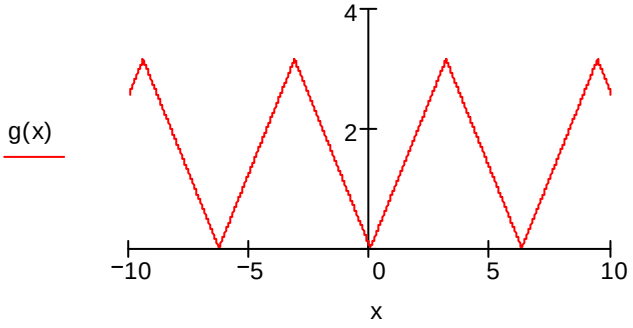


Noch komplizierter wird es, wenn - wie bei einer "**Dreiecksschwingung**", Fallunterscheidungen eingebaut werden müssen. Das soll auf 2 Arten demonstriert werden:

A. Periodische Fortsetzung der Grundfunktion f(x)

$$f(x) := \begin{cases} x & \text{if } 0 \le x < \frac{T}{2} \\ -x + 2 \cdot \pi & \text{if } \frac{T}{2} \le x < T \end{cases}$$

$g(x) := f\left(x - T \cdot \text{floor}\left(\frac{x}{T}\right)\right)$



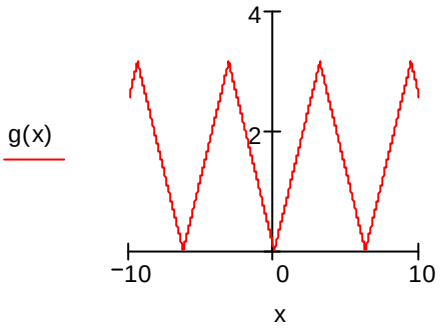
B.Berechnung der Funktion in Abhängigkeit von x

$$g(x) := \begin{cases} z \leftarrow \text{floor}\left(\dfrac{x}{\frac{T}{2}}\right) \\ x - z \cdot \pi \quad \text{if} \quad \dfrac{z}{2} = \text{floor}\left(\dfrac{z}{2}\right) \\ -(x - z\pi) + \pi \quad \text{otherwise} \end{cases}$$

z speichert, wie oft T/2 = π in x (ganzzahlig) enthalten ist!

... wenn z gerade ist

... andernfalls (z ist ungerade!)



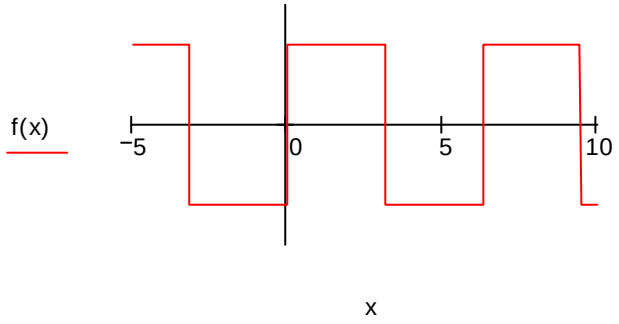
Am Beispiel einer **Rechtecksschwingng** soll die letzte Variante noch etwas anders mit Hilfe der Modulofunktion formuliert werden:

$$f(x) := \begin{cases} z \leftarrow \dfrac{x}{\frac{T}{2}} \\ \text{return } 1 \quad \text{if} \quad \text{mod}(\text{floor}(z), 2) = 0 \\ \text{return } -1 \quad \text{otherwise} \end{cases}$$

$x := -5, -4.99 .. 10$

... z ist gerade

z ist ungerade



Schritt 4: Beispiel für Textausgabe (F-Test: "signifikant" oder "nicht signifikant")

zurück zum Inhalt

Ein Programm kann als Ergebnis auch einen Text liefern. Ein Text bzw. "String" wird dabei zwischen Anführungszeichen ("....") eingeschlossen.

Beispiel: Es soll überprüft werden, ob sich die Standardabweichungen s1 bzw. s2 zweier Meßserien von n1 bzw. n2 Messungen "signifikant" unterscheiden. Dies unter der Berücksichtigung einer Irrtumswahrscheinlichkeit α (zweiseitiger Test).

Dazu wird der F-Test durchgeführt so durchgeführt, dass nur das Endergebnis in Textform ("signifikant" bzw. "nicht signifikant") ausgegeben wird.
Vorteil dieser Methode: Der Benützer der Funktion braucht sich nicht mit für ihn eventuell lästigen "mathematischen Ballast" beschäftigen.
Nachteil dieser Methode: die berechneten Größen sind außerhalb des Programmes (etwa für eine grafische Darstelllung etc.) NICHT verwendbar!

```
F_Test(α, s1, s2, n1, n2) :=
    Fpr ← if s1 > s2
            f1 ← n1 − 1
            f2 ← n2 − 1
            s1^2 / s2^2
          otherwise
            f1 ← n2 − 1
            f2 ← n1 − 1
            s2^2 / s1^2
    Fkrit ← qF(1 − α/2, f1, f2)
    return "Signifikant!" if Fpr > Fkrit
    return "Nicht signifikant" otherwise
```

Fallunterscheidung:

Die Prüfgröße wird so gebildet, dass sie größer oder gleich 1 ist, damit nur "einseitig" nach oben mit dem kritischen Wert verglichen werden muß.

Das Beispiel ließe sich (z.B. mit Verwendung der max- bzw. min-Funktion) eleganter lösen - jedoch sollte hier auch die Verwendung des if - otherwise -Operators demonstriert werden.

Berechnung der kritischen F-Größe mittels der eingebauten "Quantill-Funktion" qF (zweiseitiger Test!)

F_Test(0.05, 4.3, 2.1, 10, 10) = "Signifikant!"

F_Test(0.05, 2.1, 4.2, 10, 10) = "Nicht signifikant"

Schritt 5: Beispiele für Fehlererkennung / Fehlerausgabe

zurück zum Inhalt

Das "Abfangen" von möglichen Fehlern ist eine wichtige Aufgabe beim Programmieren. Mathcad erlaubt es, Fehlermeldungen zu "definieren". Das folgende (triviale) Beispiel zeigt die verschiedenen möglichen Techniken

```
f(x) := Fehler("x muß ungleich 0 sein") if x = 0
        1/x
```

f(0) = ▮ f(2) = 0.5

```
f(x) := Fehler("x muß ungleich 0 sein") on error 1/x
```

f(0) = ▮ f(2) = 0.5

```
f(x) := "x muß ungleich 0 sein" on error 1/x
```

f(0) → "x muß ungleich 0 sein"

f(2) = 0.5

Im folgenden, etwas komplizierteren Beispiel zur Berechnung der Kombinationen "n über k" werden verschiedene Fehlererkennungen durchgeführt:
* n UND k müssen ganze Zahlen sein
* n muß größer als k sein
* Die Berechnung über die "Fakultätsformel" führt (zumindest bei numerischer Auswerung mit "=") bei größeren Zahlen zu Problemen. Mit der "on error"-Funktion wird dieser Fehler abgefangen (allerdings wäre hier natürlich gleich die links stehende Formulierung allein ausreichend gewesen)

$$C(n,k) := \begin{cases} \text{Fehler("n und k müssen ganz sein")} \quad \text{if} \quad n \neq \text{floor}(n) \vee k \neq \text{floor}(k) \\ \text{(Fehler("n muß größer als k sein"))} \quad \text{if} \quad (n < k) \\ \prod_{i=1}^{k} \left(\frac{n-i+1}{i} \right) \quad \text{on error} \quad \frac{n!}{k! \cdot (n-k)!} \end{cases}$$

$C(5,4) = 5$ $C(5,6) =$ ∎ $C(3.7,2) =$ ∎ $C(6,6) = 1$

$C(10000,100) = 6.521 \times 10^{241}$

Schritt 6: Verwendung von Schleifen

zurück zum Inhalt

Das Prinzip werde zunächst an der einfachen Summation der natürlichen Zahlen von 1 bis N erklärt (Erinnerung an die Anekdote von C.F.Gauss)

$N := 100$ $\sum_{i=1}^{N} i \rightarrow 5050$

Dies soll auf 2 Arten als "Programm" formuliert werden:

```
summe(N1, N2) := | sum ← 0
                 | for i ∈ N1 .. N2
                 |   sum ← sum + i
```

Notwendige Initialisierung der lokalen Variablen "sum"

Eingabe: "for" anklicken, Platzhalter ausfüllen.

summe(1, 100) = 5050

summe(100, 1) = 5050

```
summe(N1, N2) := | sum ← 0
                 | i ← min(N1, N2)
                 | while i ≤ max(N1, N2)
                 |     | sum ← sum + i
                 |     | i ← i + 1
                 | return sum
```

Die Formulierung mittels "while" Schleife ist deswegen hier wesentlich umständlicher, weil die "for"-Schleife als eine abgekürzte while- Schleife für besondere Fälle (Anfang und Ende von vornherein bekannt) gedeutet werden kann.

"while" sollte man am besten mit "solange" übersetzen!

summe(1, 100) = 5050

summe(100, 1) = 5050

Im folgenden Beispiel werden (in Anlehnung an den Euklidischen Algorithmus) verschachtelte Schleifen zur Berechnung des größten gemeinsamen Teilers zweier Zahlen verwendet. Zur besseren Übersicht erfolgt hier nicht die an sich notwendige Absicherung, dass a und b ganze Zahlen sind! ("Übungsaufgabe")

```
ggT(a, b) := | while a ≠ b
             |     | while a > b
             |     |   a ← a − b
             |     | while b > a
             |     |   b ← b − a
             | return a
```

$ggT(80, 28) = 4$ $ggT(4, 12) = 4$ $ggT(13, 17) = 1$ $ggT(27, 27) = 27$

Schritt 7: Ein komplexeres Beispiel: Das Newtonverfahren

zurück zum Inhalt

Diese Beispiel wurden gewählt, weil es sich um ein bekanntes numerische Verfahren handelt. Es ist einerseits nicht zu lang sind, benötigt aber die meisten Programmierelemente, welche Mathcad zur Verfügung stellt.

Newtonverfahren:
x ... Schätzwert (Anfangswert) für die Newton-Iteration (Nullstellensuche der Funktion f(x))
f ... Name einer zuvor definierten Funktion
f_x ... Ableitungsterm der Funktion f(x)

```
newton(x, f, fx) :=
    i ← 0
    while |f(x)| > 10^-6
        i ← i + 1
        break if i ≥ 10
        return "Ableitung ist 0" if |fx(x)| < 10^-6
        x ← x - f(x)/fx(x) otherwise
    return "Zuviele Iterationen" if i ≥ 10
    return x otherwise
```

Festlegung der erlaubten Abweichung

nach 10 Iterationsschritten Abbruch!

Absicherung, falls waagrechte Tangente

Iterationsformel

"return" hier nicht unbedingt erforderlich, aber übersichtlicher!

Beispiel für Anwendung des Programmes

$f(x) := \sin(x) + x - 1$ $x := 0, 0.01 .. 5$

$f_x(x) := \frac{d}{dx} f(x)$

$x_1 := \pi - 0.4$

$x_{ns} := newton(x_1, f, f_x)$ $x_{ns} =$ "Zuviele Iterationen"

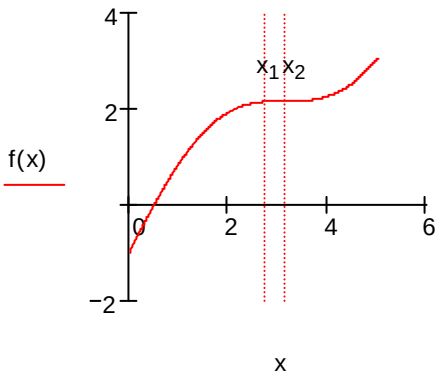
$x_2 := \pi$

$x_{ns} := newton(x_2, f, f_x)$ $x_{ns} =$ "Ableitung ist 0"

$x_3 := 2$

$x_{ns} := newton(x_3, f, f_x)$ $x_{ns} = 0.51097$

$f(x_{ns}) = -1.352 \times 10^{-13}$



Schritt 8: Beispiel zur Rekursion (Fibonacci-Folge)

zurück zum Inhalt

Rekursion bedeutet, dass eine Funktion auch sich selbst aufrufen kann. Damit sind in bestimmten Fällen sehr elegante Formulierungen von Programmen möglich, allerdings mit dem Preis eines langsameren Ablaufes des Programmes. Beliebte Beispiele für Rekursionen im Schulnunterricht sind beispielsweise:
* Die Fibonacci-Zahlenfolge
* Die Programmierung des Algorithmus zum "Turm von Hanoi"
* Die Koch´sche Schneeflockenkurve und andere Fraktale

Da das Grundprinzip immer das Gleiche ist, wird hier die Rekursion am Beispiel der bekannten Fibonacci-Folge geschildert. Diese entsteht, indem (ausgehend von ZWEI Startwerten) immer die LETZTEN 2 Glieder der Folge zusammengezählt werden. Als Startwerte werden meist (gemäß der Urform der Fibonacci-Folge) 1 und 1 gewählt.

Bei jeder Rekuriosn muß darauf geachtet werden, dass
*** die Startwerte bzw. der "Einstieg" richtig ist.**
*** ein Ende der Rekursion durch eine entsprechende Ausstiegsbedingung gegeben ist.**

$Startwert := 1$

$$Fib(n) := \begin{cases} \text{Fehler("Parameter muß größer/gleich 0 sein")} & \text{if } n < 0 \\ \text{Fehler("Parameter muß ganzzahlig sein")} & \text{if } n \neq \text{floor}(n) \\ Startwert & \text{if } n = 0 \vee n = 1 \\ Fib(n-1) + Fib(n-2) & \text{otherwise} \end{cases}$$

$n := 0 .. 15$

n =	Fib(n) =	$\frac{Fib(n+1)}{Fib(n)}$ =
0	1	1
1	1	2
2	2	1.5
3	3	1.6666666667
4	5	1.6
5	8	1.625
6	13	1.6153846154
7	21	1.619047619
8	34	1.6176470588
9	55	1.6181818182
10	89	1.6179775281
11	144	1.6180555556
12	233	1.6180257511
13	377	1.6180371353
14	610	1.6180327869
15	987	1.6180344478

Die Formulierung des gleichen Problems mit Hilfe von Schleifen zeigt die Stärken des rekursiven Programmaufbaues (elegante und verständliche Formulierung) - aber auch die Schwächen (Laufzeit!), wenn beispielsweise - wie hier - gezeigt werden soll, dass der Quotient zweier aufeinanderfolgender Glieder der Fibonacci-Reihe gegen einen Grenzwert strebt (es ist dies die Zahl ϕ des "Goldenen Schnittes").

```
Fibon(n) := | wert ← 1  if  n ≤ 1
            | otherwise
            |     h1 ← 1
            |     h2 ← 1
            |     for  k ∈ 2 .. n
            |         tmp ← h1 + h2
            |         h1 ← h2
            |         h2 ← tmp
            |     wert ← tmp
            | wert
```

$n := 0 .. 15$

n =	Fibon(n) =	$\frac{Fibon(n+1)}{Fibon(n)}$ =
0	1	1
1	1	2
2	2	1.5
3	3	1.6666666667
4	5	1.6
5	8	1.625
6	13	1.6153846154
7	21	1.619047619
8	34	1.6176470588
9	55	1.6181818182
10	89	1.6179775281
11	144	1.6180555556
12	233	1.6180257511
13	377	1.6180371353
14	610	1.6180327869
15	987	1.6180344478