



F3 — Berechenbarkeit und Komplexität

Berndt Farwer

Fachbereich Informatik

AB „Theoretische Grundlagen der Informatik“ (TGI)

Universität Hamburg

farwer@informatik.uni-hamburg.de



Zielgruppe

1. Informatik (3. Semester)
2. Wirtschaftsinformatik (3. Semester)
3. Allgemeine Ingenieurwissenschaften,
Informatikingenieurwesen und TECMA
(5. Semester)

Wichtig:

Beginn 12:15

Ende 13:45



Übungsgruppen

... gibt es nur für Wirtschaftsinformatiker und Studierende der TU-Harburg!

Mo 12–13 C-221

Michael Köhler

Mo 13–14 C-221

Michael Köhler

Di 11–12 Binderstr. 22/54 (nur TU) Berndt Farwer

Do 12–13 C-221

Daniel Moldt

Do 13–14 C-221

Daniel Moldt

Do 16–17 F-535

Mark-Oliver Stehr

Do 17–18 F-535

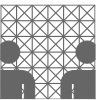
Mark-Oliver Stehr



Scheinkriterien

Bedingungen für die Ausstellung eines Scheines:

- 50% der erreichbaren Punkte
- regelmäßige, aktive Teilnahme an den Übungsgruppen
- Vorrechnen an der Tafel
- max. zweimaliges unentschuldigtes Fehlen



Das Skript zur Vorlesung ist erhältlich:

1. **gedruckt** über

- das Sekretariat von TGI (C-218)
- die Übungsgruppe

2. **elektronisch**

- <http://www.informatik.uni-hamburg.de>
- —→ Studium und Prüfungen —→ Skripte
... **dort auch Übungsaufgaben und
Musterlösungen sowie aktuelle
Ankündigungen!**

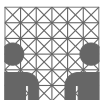


Ägyptische Multiplikation

... ein Zahlenbeispiel: $231 \cdot 101 = 23331$

231	101	
115	202	
57	404	
28	808	(zu streichen)
14	1616	(zu streichen)
7	3232	
3	6464	
1	12928	

Diese Methode ist korrekt! (Beweis mit Hilfe der Binärdarstellungen der Zahlen)



Ägyptische Multiplikation (2)

Beispielrechnung: $[23]_{10} \cdot [11]_{10}$

$$[23]_2 = 10111 \quad 1011 = [11]_2$$

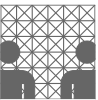
$$[11]_2 = 1011 \quad 10110 = [22]_2$$

$$[5]_2 = 101 \quad 101100 = [44]_2$$

$$[2]_2 = 10 \quad 1011000 = [88]_2$$

$$[1]_2 = 1 \quad 10110000 = [176]_2$$

$$11111101 = [253]_2$$



Ägyptische Multiplikation (3)

Warum ist das Verfahren für die Informatik interessant?

1. **ganzzahlige Division durch 2:**
letztes Bit abschneiden
2. **ganzzahlige Multiplikation mit 2:**
0 anhängen

Diese elementaren Operationen sind leicht zu implementieren!



Korrektheit des Verfahrens

- Beispiele können höchstens **Fehler** im Verfahren aufdecken
- Korrektheit muß **bewiesen** werden, z.B. durch
 - Induktionsbeweis
 - Widerspruchsbeweis
 - andere mathematische Verfahren



Problemtypen

Je nach Aufgabenstellung lassen sich verschiedene Grundtypen von Problemen unterscheiden:

1. **Entscheidungsprobleme**
2. **Suchprobleme**
3. **Optimierungsprobleme**
4. **Abzählungsprobleme**
5. **Anzahlprobleme**



Entscheidungsprobleme

... zum Beispiel:

Gegeben:	Eine natürliche Zahl $n \in \mathbb{N}$.
Gesucht:	Antwort auf die Frage: Ist n eine Primzahl?
Antwort:	JA oder NEIN

Problem Π ist *Entscheidungsproblem*, gdw. Lösungsraum $\mathcal{L}(\Pi)$ besteht aus genau zwei Elementen und jede Instanz $I \in \mathfrak{I}(\Pi)$ hat genau eine Lösung.

Z.B. $\mathcal{L}(\Pi) = \{0, 1\}$.



Suchprobleme

... zum Beispiel:

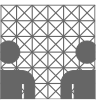
Gegeben:	ungerichteter Graph $G := (V, E)$ mit $E \subseteq V \times V$ und Knoten $v_1, v_2 \in V$.
Gesucht:	ein Weg von v_1 nach v_2
Antwort:	Kantenfolge eines Pfades oder „Es gibt keinen!“



Optimierungsprobleme

... zum Beispiel:

Gegeben:	gerichteter, bewerteter Graph $G := (V, E)$ mit $E \subseteq V \times \mathbb{N} \times V$ und Knoten $v_1, v_2 \in V$.
Gesucht:	ein günstigster Weg von v_1 nach v_2
Antwort:	Kantenfolge eines Pfades (evtl. mit Kosten) oder „Es gibt kei- nen!“



Abzählungsprobleme

... zum Beispiel:

Gegeben:	endliche Menge von Objekten
Gesucht:	alle binären Suchbäume für diese Objekte
Antwort:	Aufzählung der binären Suchbäume



Anzahlprobleme

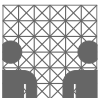
... zum Beispiel:

Gegeben:	zwei Klammersymbole [und]
Gesucht:	Wieviele korrekt geklammerte Terme mit $2n$ Klammersymbolen gibt es?
Antwort:	Eine Zahl.





Der Begriff des Algorithmus und die Turing-Maschine



Algorithmus

... einige Algorithmus-Definitionen:

Ein **Algorithmus** liegt genau dann vor, wenn gegebene Größen, auch **Eingabegrößen**, **Eingabeinformationen** oder **Aufgaben** genannt, auf Grund eines Systems von **Regeln**, Umformungsregeln, in andere Größen, auch **Ausgabegrößen**, **Ausgabeinformationen** oder **Lösungen** genannt, umgeformt oder umgearbeitet werden.

Kleine Enzyklopädie MATHEMATIK, 1968



Algorithmus (2)

Ein **Algorithmus** ist eine präzise, d.h. in einer festgelegten Sprache abgefaßte, endliche Beschreibung eines allgemeinen Verfahrens unter Verwendung ausführbarer elementarer (Verarbeitungs-)Schritte.

Bauer, Goos, Informatik I, 1991

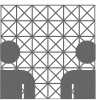
Wichtig: Es gibt *terminierende* und *nicht terminierende* Algorithmen!



Algorithmus (3)

Ein Algorithmus soll also

- schrittweise arbeiten (**Diskretheit**),
- nach jedem Schritt eindeutig bestimmen, was der nächste Schritt ist (**Determiniertheit**),
- einfache Schritte enthalten (**Elementarität**).
- nach endlichen vielen Schritten zu einer Lösung führen (**Konklusivität**),
- auf eine hinreichend große Klasse von Instanzen anwendbar sein (**Generalität**).
- sich mit endlichen Mitteln beschreiben lassen (**endliche Beschreibbarkeit**).



Berechenbarkeit

Bereits vor der Prägung des Algorithmusbegriffs:
Suche nach formaler Definition von
Berechenbarkeit.

Wichtige Observationen:

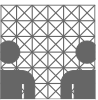
1. Unlösbare Probleme werden auch bei Verwendung immer schnellerer Rechner unlösbar bleiben.
2. Unter den lösbaren Problemen: Existenz schwer-lösbarer Probleme. Auch schnelle Computer können daran nichts ändern.



Beschreibungsformen für Alg.

- verbal
- programmiersprachenähnlich
- graphisch
- mathematisch

... aber wie soll man einen Algorithmus mathematisch notieren?!



Beispiel

Summe der ersten n natürlichen Zahlen:

Gegeben: $n \in \mathbb{N}$
Gesucht: $s = \sum_{i=1}^n i$
Antwort: ?

(a) verbale Beschreibung:

Beginne mit $Summe := 0$. Addiere sukzessive zu $Summe$ die Zahlen 1 bis n . Am Ende enthält $Summe$ das gesuchte Resultat.



Beispiel (2)

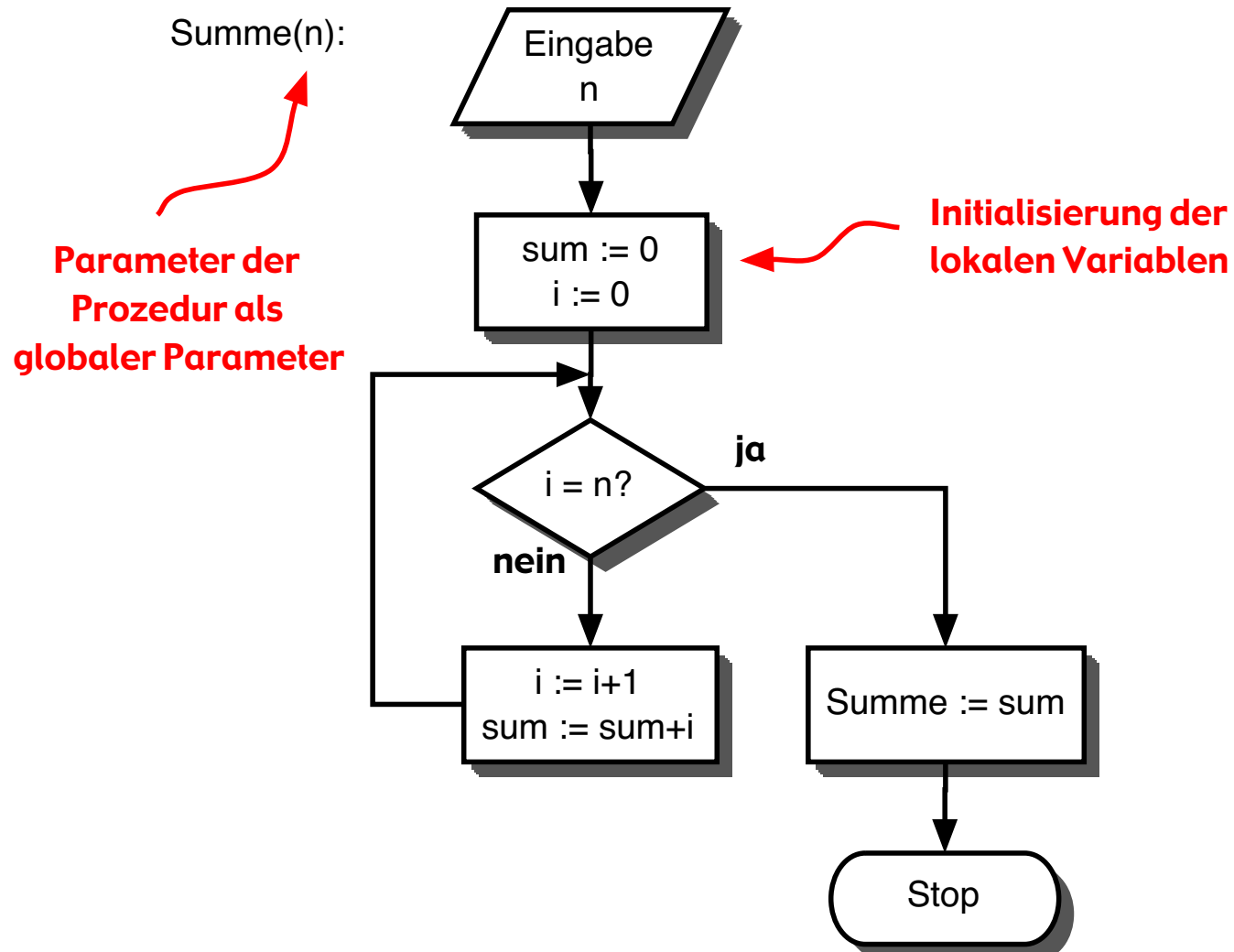
(b) programm-ähnliche Notation:

```
function Summe( $n$ )  
  {berechnet die Summe der natürlichen  
   Zahlen von 1 bis  $n$ }  
  sum  $\leftarrow$  0  
  for  $i \leftarrow 1$  to  $n$  do sum  $\leftarrow$  sum +  $i$   
  return sum
```



Beispiel (3)

(c) graphische Notation:



Die Gaußsche Formel

Theorem: Für beliebiges $n \in \mathbb{N}$ gilt:

$$1 + 2 + 3 + \dots + n = \binom{n+1}{2} = \frac{(n+1) \cdot n}{2}.$$

Algorithmus: Summe der ersten n natürlichen Zahlen ist $\binom{n+1}{2}$.

... das sieht viel einfacher aus, aber ist es auch korrekt?

→ Das muß erst bewiesen werden!



Beweis der Gaußschen Formel

(a) direkte Methode von Gauß:

$$\begin{aligned} 2s &= 2 \cdot \sum_{i=1}^n i \\ &= 1 + 2 + \dots + (n-1) + n + \\ &\quad n + (n-1) + \dots + 2 + 1 \\ &= (n+1) \cdot n \end{aligned}$$

Also:

$$s = \frac{n(n+1)}{2}$$



Beweis der Gaußschen Formel (2)

(b) mit vollständiger Induktion:

Verankerung:

$$n = 0 \longrightarrow \sum_{i=0}^0 i = 0 = \frac{0 \cdot (0 + 1)}{2}$$

Induktionsannahme:

Die Gaußsche Formel gilt für festes, beliebiges $m \in \mathbb{N}$.

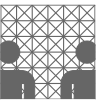


Beweis der Gaußschen Formel (2)

Induktionsschritt:

$$\begin{aligned}\sum_{i=0}^{m+1} i &= \sum_{i=0}^m i + (m+1) \\ &= \frac{m \cdot (m+1)}{2} + (m+1) \\ &= \frac{m \cdot (m+1) + 2 \cdot (m+1)}{2} \\ &= \frac{(m+1)(m+2)}{2}\end{aligned}$$

Somit ist die Annahme bewiesen!



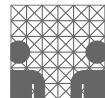
Problemlösung

... besteht aus:

- Problem, vorhandene Eingaben und gewünschte Ausgaben korrekt erfassen
- Algorithmus finden, der das Problem löst
- Algorithmus als korrekt nachweisen
 - mittlere/schlechteste Laufzeit und Speicherbedarfe ermitteln
 - Algorithmus effizient implementieren



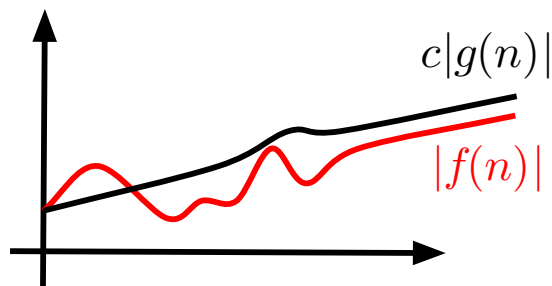
Die Landau- oder O-Notation



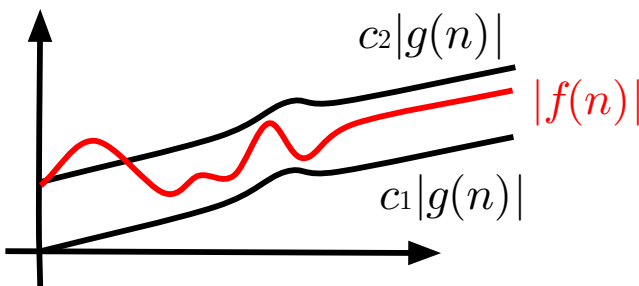
Anschaulich

Wir betrachten das Wachstum von Funktionen:

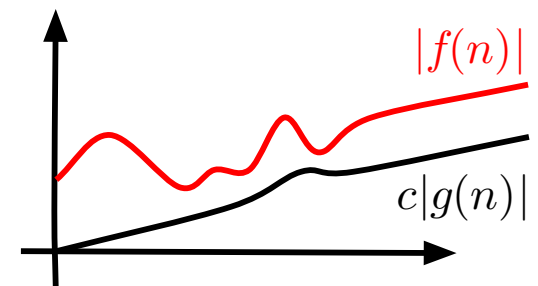
Auf *Paul Bachmann* (1894) geht die von E. Landau popularisierte Notation zurück.



$$f(n) \in O(g(n))$$



$$f(n) \in \Theta(g(n))$$



$$f(n) \in \Omega(g(n))$$



O-Notation

Definition:

Eine Funktion $f : \mathbb{N} \rightarrow \mathbb{R}$
wächst mit der Ordnung $g(n)$ bzw. g ,
geschrieben

$$f(n) \in O(g(n)) \text{ bzw. } f \in O(g),$$

falls $g : \mathbb{N} \rightarrow \mathbb{R}$ eine Funktion ist und eine
Konstante $c \in \mathbb{R}^+ := \{x \in \mathbb{R} \mid x > 0\}$ existiert,
so daß

$$|f(n)| \leq c \cdot |g(n)|$$

für alle, bis auf endlich viele $n \in \mathbb{N}$ gilt.



O-Notation (2)

Etwas knapper notiert liest sich das so:

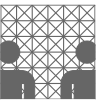
$$O(g) =$$

$$\{f : \mathbb{N} \rightarrow \mathbb{R} \mid (\exists c \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N})(\forall n \geq n_0) \\ [|f(n)| \leq c |g(n)|]\}.$$

... die Menge aller Funktionen, für die g (mit einem konstanten Faktor) eine obere Schranke ist.

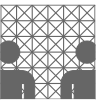
(„ f wächst nicht schneller als g “)

Mit $O(f)$ bezeichnen wir also eine Menge von Funktionen.



Zusammenfassung

- Die O-Notation ist asymptotische Notation. Sie spiegelt das Verhalten einer Funktion nur für große n korrekt wider. $f(n) \in O(g(n))$ bedeutet:
 - $g(n)$ und $f(n)$ haben für große n ein ähnliches Wachstumsverhalten.
 - Es ist $\frac{|f(n)|}{|g(n)|} \leq c$ für alle $n \geq n_0$ mit $g(n) \neq 0$.
 - Es gilt, falls $a_k \neq 0$:
$$O(a_k n^k + a_{k-1} n^{k-1} + \dots + a_1 n + a_0) = O(n^k)$$
 - $O(f(n) + g(n)) = O(\max(|f(n)|, |g(n)|))$.



Beispiele zur O-Notation

Beispiele:

Seien f und g definiert durch:

$$f(n) := 12n^4 - 11n^3 + 1993 \text{ und } g(n) := 7n^3 - n.$$

Dann ist $g \in O(f)$ aber nicht $f \in O(g)$.

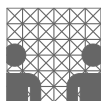
Auch gilt

$$f \in O(n^4)$$

und

$$f \in O(7028060n^4 - 1948).$$

$$1000x \in O(x^2 - x)$$



Die Funktionsklassen $\Omega(g)$

Definition:

$$\Omega(g) =$$

$$\{f : \mathbb{N} \rightarrow \mathbb{R} \mid$$

$$(\exists c \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N})(\forall n \geq n_0)[c|g(n)| \leq |f(n)|]\}$$

... die Menge aller Funktionen, für die g (mit einem konstanten Faktor) eine untere Schranke ist.

(„ f wächst nicht langsamer als g “)



Funktionsklassen $\Theta(g)$

Definition:

$$\begin{aligned}\Theta(g) = \\ \{f : \mathbb{N} \rightarrow \mathbb{R} \mid \\ (\exists c_1, c_2 \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N}) \\ (\forall n \geq n_0)[c_1 |g(n)| \leq |f(n)| \leq c_2 |g(n)|]\}.\end{aligned}$$

„ f wächst etwa so schnell wie g “



Beispiel zu $\Theta(g)$

Beispiel: Um zu beweisen, daß

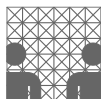
$$\frac{1}{2}n^2 - 3n \in \Theta(n^2) \quad n \rightarrow \infty$$

muß man solche $c_1 > 0, c_2, n_0$ finden, daß

$$c_1 n^2 \leq \left| \frac{1}{2}n^2 - 3n \right| \leq c_2 n^2 \quad \text{für } n \geq n_0$$

$$c_1 \leq \left| \frac{1}{2} - \frac{3}{n} \right| \leq c_2 \quad \text{für } n \geq n_0$$

Lösungen: $c_1 \leq \frac{1}{14}, \quad c_2 \geq \frac{1}{2}, \quad n \geq 7$



Weitere Funktionsklassen

Definition:

$$o(g) = \{f : \mathbb{N} \rightarrow \mathbb{R} \mid (\forall c \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N}) \\ (\forall n \geq n_0)[|f(n)| \leq c |g(n)|]\}$$

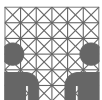
„ f wächst langsamer als g “

Beispiel: $\ln(x) \in o(\sqrt{x})$ und $\sqrt{x} \in o(x)$

Definition:

Es sei $f \in \omega(g)$ genau dann, wenn $g \in o(f)$

„ f wächst schneller als g “



Zusammenhänge

... grundlegende Beziehungen zwischen Θ -, O - und Ω -Notation:

$$f(n) \in \Theta(g(n)) \iff f(n) \in O(g(n)) \text{ und } f(n) \in \Omega(g(n)).$$

Eine Auswahl von Regeln zur Manipulation von O -Ausdrücken:

$$n^m \in O(n^{m'}) \quad \text{falls } m \leq m'$$

$$f(n) \in O(f(n))$$

$$cO(f(n)) = O(f(n))$$

$$O(O(f(n))) = O(f(n))$$

$$O(f(n)) + O(g(n)) = O(|f(n)| + |g(n)|)$$

$$O(f(n))O(g(n)) = O(f(n)g(n))$$



kanonische Erweiterung

Man schreibt $f(n) + O(g(n))$ für

$$\{g' \mid g'(n) := f(n) + h(n) \text{ und } h \in O(g)\}$$

...zum Beispiel:

$$\sum_{k=0}^n (e + O(k)) \quad e, k \in \mathbb{R}$$

$$e + O(k) = \{f : \mathbb{N} \rightarrow \mathbb{R} \mid \exists c : |f(n, k)| \leq c \cdot k\}$$

Also: $\sum_{k=0}^n (e + O(k))$ enthält alle Funktionen
der Form: $\sum_{k=0}^n e + \sum_{k=0}^n f(n, k)$

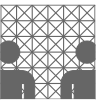


Abschätzung

Wegen $|f(n, k)| \leq c \cdot k$ folgt:

$$\begin{aligned} & e \cdot (n + 1) + |f(n, 0)| + |f(n, 1)| + \cdots + |f(n, n)| \\ & \leq e \cdot (n + 1) + c \cdot 0 + c \cdot 1 + \cdots + c \cdot n \\ & = e \cdot (n + 1) + c \cdot \frac{n(n+1)}{2} \\ & = c \cdot \frac{n^2}{2} + \left(\frac{c}{2} + e\right) \cdot n + e \\ & \leq d \cdot n^2 \end{aligned}$$

Also: $\sum_{k=0}^n (e + O(k)) \subseteq O(n^2)$,
da die linke Seite eine Menge von Funktionen
beschreibt!



log-Funktion

Definition: Wir schreiben $\log(n)$ anstelle des oft üblichen $\log n$, außer in der Variante $(\log n)$ statt $(\log(n))$, und meinen damit:

$$\log(n) := \begin{cases} 1 & , \text{ falls } n \leq 1 \\ \lceil \log_2(n) \rceil & , \text{ sonst.} \end{cases}$$

... die Länge der Binärdarstellung einer natürlichen Zahl.

Anzahl der Speicherzellen/Schritte ist eine natürliche Zahl.



...jetzt haben wir die Grundlagen zum Einordnen von Funktionen (Schränken).

ABER: was wollen wir eigentlich abschätzen?

Zeit und Platz müssen formalisiert werden.

—→ Turing-Maschine zur *Vereinheitlichung des Zeitbegriffes* (**Rechenschritte**, **Konfigurationsübergänge**) und des *Speicherbedarfes* (**Bandzellen**).

...im Skript: Beispiel ggt-Berechnung



Ausblick

