RUPRECHT-KARLS-
UNIVERSITÄT
HEIDELBERG

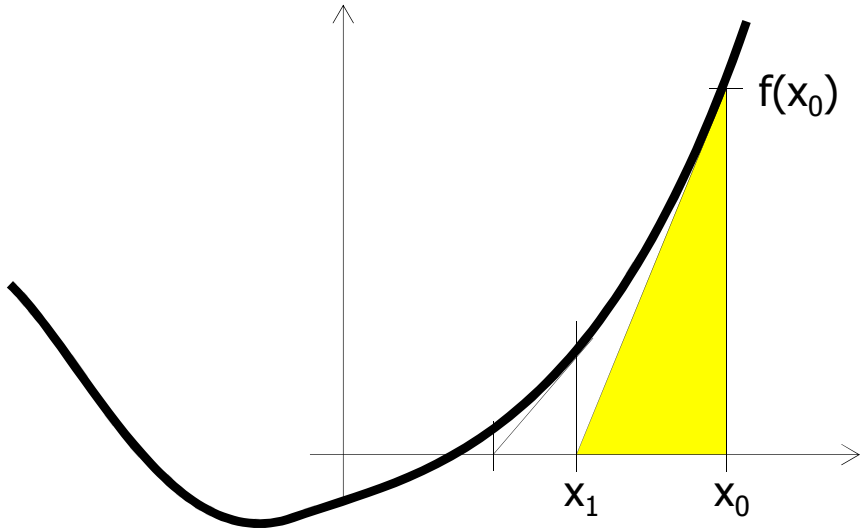
Ein Paar Zusatzfolien zum C++ Programmierkurs

Prof. Dr. P. Fischer

Lehrstuhl für Schaltungstechnik und Simulation
Technische Informatik der Uni Heidelberg

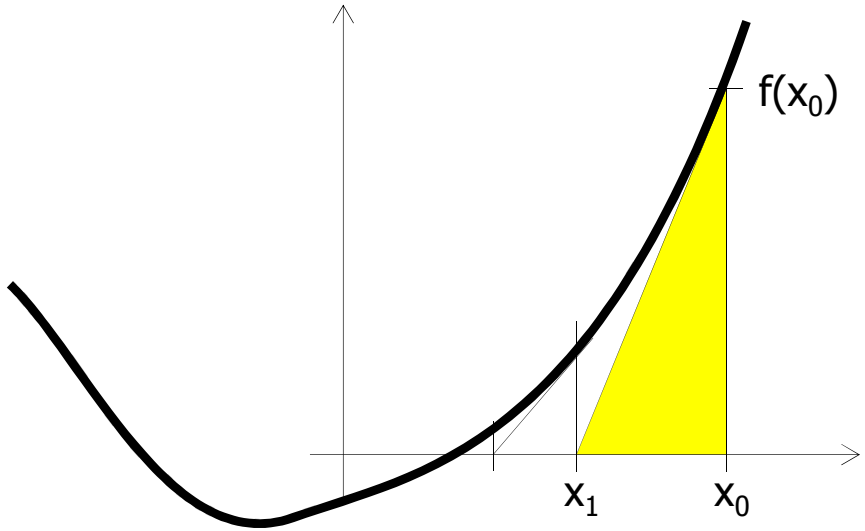
Newton Iteration

- Suche Nullstelle einer Funktion f(x)
- Idee:
  - Beginne mit Näherungswert x_0
  - Lege Tangente an Funktion bei x_0, $f(x_0)$
  - Neuer Wert x_0 (oder x_1) ist Schnittpunkt der Tangente mit x-Achse



Newton Iteration

- Suche Nullstelle einer Funktion f(x)
- Idee:
  - Beginne mit Näherungswert x_0
  - Lege Tangente an Funktion bei x_0, $f(x_0)$
  - Neuer Wert x_0 (oder x_1) ist Schnittpunkt der Tangente mit x-Achse



- Für die Steigung des Dreiecks gilt:
  - $f'(x_n) = \Delta y/\Delta x = f(x_n) / (x_n - x_{n+1})$
  - $x_{n+1} = x_n - f(x_n) / f'(x_n)$ (Offenbar ist $x_{n+1} = x_n$ für $f(x_n)=0$)

Wurzel berechnen

- Um die r-te Wurzel aus c finden, suchen wir z.B. die Nullstelle der Funktion
  $f(x) = x^r - c$

- Die Iterationsvorschrift lautet damit:
  - $x_{n+1} = x - f(x) / f'(x) = x - (x^r - c)/(r\, x^{r-1})$ $\quad (x = x_n)$

- Für r=2:
  - $x_{n+1} = x - (x^2 - c)/(2\, x) = x/2 + c/(2\, x)$

- Für c=2: (also für die Berechnung von $\sqrt{2}$):
  - $x_{n+1} = x/2 + 1/x$

Optimierung

- Die Iterationsvorschrift $x_{n+1} = x/2 + 1/x$ enthält eine Division durch x. Das ist langsam!

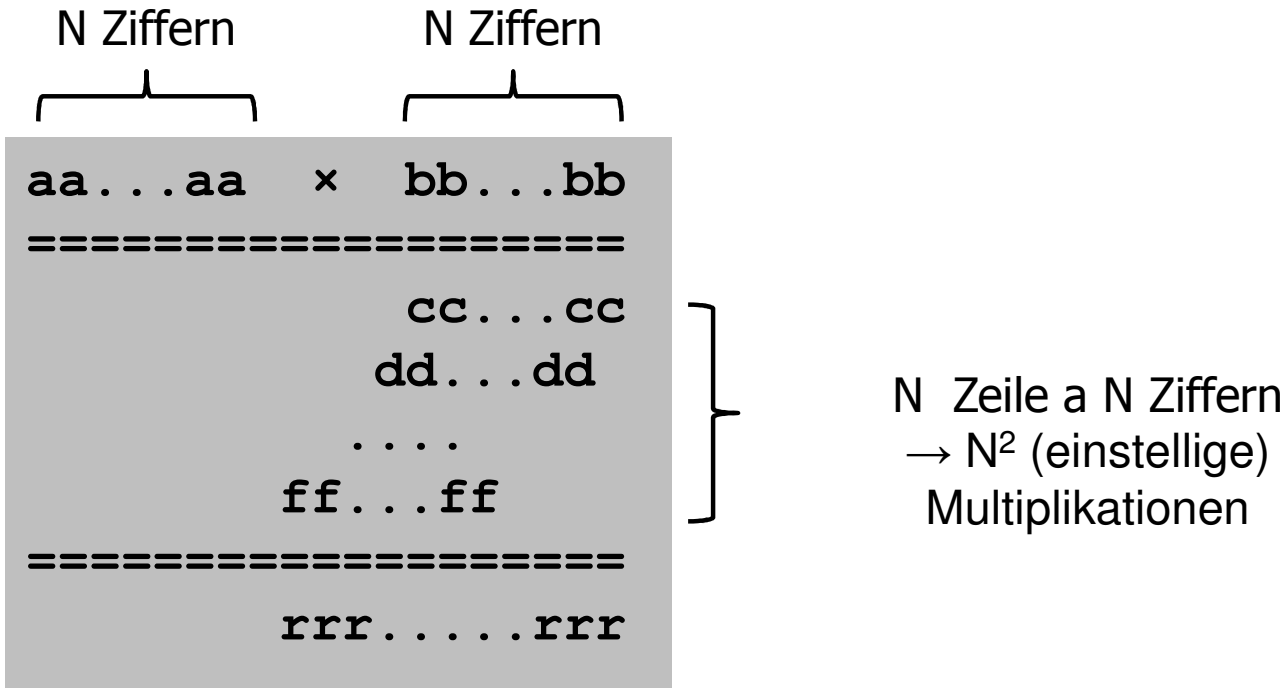
- Effizienter sucht man zunächst $1/\sqrt{2}$:
  - $f(x) = 1/x^2 - 2$
  - $f'(x) = -2/x^3$
  - $x_{n+1} = x - f(x) / f'(x) = x + (1/x^2 - 2)\, x^3/2 = x + x/2 - x^3$
    $= x\,(3/2 - x^2)$
- Das geht OHNE Division.

- Dann berechnet man $\sqrt{2} = 2 \times 1/\sqrt{2}$

KARATSUBA MULTIPLIKATION

Wie ist der Aufwand für eine Multiplikation ?

- Schul - Multiplikation von zwei N-stelligen Zahlen A, B:



- Aufwand (Maß hier: Anzahl einstellige Multiplikationen):

$$O_{\text{Schul-Mult}}(N) \sim N^2$$

Kann man sparen ?

- Versuch: Zerlege A und B in ‘hohe’ und ‘niedrige’ Hälften A_H/A_L, die jeweils nur N/2 Stellen lang sind.

- $A \times B = (A_H \times 10^{N/2} + A_L) \times (B_H \times 10^{N/2} + B_L)$
  $= \mathbf{A_H B_H} \times 10^N + (\mathbf{A_L B_H} + \mathbf{B_L A_H})\, 10^{N/2} + \mathbf{A_L B_L}$

- Man benötigt 4 Multiplikationen von Zahlen mit N/2 Stellen

- Insgesamt:

  $O_{Zerlegt} = 4 \times O_{Schul\text{-}Mult}(N/2) = 4 \times (N/2)^2 = N^2$ wie vorher

- Kein Gewinn ☹

Karatsuba Multiplikation (Karatsuba 1960, russischer Math.)

- Wir zerlegen x und y in eine ‘hohe’ und eine ‘niedrige’ Hälfte
- Ausmultiplizieren ergibt (b = Basis = 10, n=halbe Stellenzahl):

$$x \cdot y = (x_h \cdot b^n + x_l) \cdot (y_h \cdot b^n + y_l) = x_h y_h \cdot b^{2n} + (x_h y_l + x_l y_h) \cdot b^n + x_l y_l.$$

- Den Klammerteil kann man umformen:

$$x_h y_l + x_l y_h = (x_h + x_l) \cdot (y_h + y_l) - x_h y_h - x_l y_l$$

- Insgesamt:

$$x \cdot y = x_h y_h \cdot b^{2n} + ((x_h + x_l) \cdot (y_h + y_l) - x_h y_h - x_l y_l) \cdot b^n + x_l y_l$$

oder

$$x \cdot y = P_1 \cdot b^{2n} + (P_3 - P_1 - P_2) \cdot b^n + P_2$$

$$P_1 = x_h \cdot y_h$$
$$P_2 = x_l \cdot y_l$$
$$P_3 = (x_h + x_l) \cdot (y_h + y_l)$$

- Man benötigt also nur 3 Produkte der Breite n.
- Rekursiv wird der Aufwand $O(n^{\log_2(3)}) = O(n^{1,585})$, nicht $O(n^2)$!!!
- (Es geht noch schneller..)

Beispiel Karatsuba Multiplikation

$$x \cdot y = P_1 \cdot b^{2n} + (P_3 - P_1 - P_2) \cdot b^n + P_2$$

$$P_1 = x_h \cdot y_h$$
$$P_2 = x_l \cdot y_l$$
$$P_3 = (x_h + x_l) \cdot (y_h + y_l)$$

- 12 × 34

- P1 = 1 × 3 = 3
  P2 = 2 × 4 = 8
  P3 = (1+2) × (3+4) = 21
- P3 – P1 – P2 = 10

- 12 × 34 = 300
  100
  8
  408

Beispiel Karatsuba Multiplikation

$$x \cdot y = P_1 \cdot b^{2n} + (P_3 - P_1 - P_2) \cdot b^n + P_2$$

$$P_1 = x_h \cdot y_h$$
$$P_2 = x_l \cdot y_l$$
$$P_3 = (x_h + x_l) \cdot (y_h + y_l)$$

- 1234 × 5678

- P1 = 12 × 56 = 672
  P2 = 34 × 78 = 2652
  P3 = (12+34) × (56+78) = 6164
- P3 – P1 – P2 = 2840

- 1234 × 5678 = 6720000
  284000
  2652
  7006652

Auslagern von Programmteilen

vector.h

```
struct Tvect {…};

float Abs(Tvect v);
Tvect operator+(..);
```

vector.cc

```
#include "vector.h"
#include <cmath>

struct Tvect {…};

float Abs(Tvect v)
{…}

Tvect operator+(..)
{…}
```

main.cc

```
#include …
#include "vector.h"

int main(void)
{
Tvect v1;
…
float f = Abs(v1);
…
}
```

```
g++ -Wall -c vector.cc
```

```
g++ -Wall -c main.cc
```

vector.o

main.o

```
g++ -o MyExe vector.o main.0
```

MyExe

Mehrfaches Inkludieren vermeinden

- Es kommt schnell vor, dass über geschachtelte **`#includes`** eine Header mehrfach ausgeführt wird.
- Dies führt dann zu ‚multiple declaration' Fehlern.
- Man vermeidet das mit Präprozessor Befehlen:

if **not** defined

Wird nur beim ersten Durchlauf ausgeführt. Dann ist der String bekannt.

vector.h

```
#ifndef some_unique_string_best_with_vectorh
#define some_unique_string_best_with_vectorh


…
Inhalt des Header files
…


#endif
```