BERGISCHE UNIVERSITÄT WUPPERTAL
FACHBEREICH C – MATHEMATIK
UND NATURWISSENSCHAFTEN

WISSENSCHAFTLICHES RECHNEN/
SOFTWARETECHNOLOGIE
Walter Krämer

Weiterführende Veranstaltung
Verifikationsnumerik

Walter Krämer

BUW, Wuppertal, Juli 2011

Ziel der Verifikationsnumerik: verifizierte numerische Ergebnisse

Gegebenenfalls:

- Existenz der Lösung
- Eindeutigkeit der Lösung
- Einschließung der Lösung (in enge Schranken)
- Sicheres Rechnen auch mit toleranzbehafteten Daten

Typische Hilfsmittel:

- Mengenarithmetik (insb. Intervallrechnung; gerichtet gerundete Gleitkommaoperationen)
- Automatische Differentiation
- Anwendung mathematischer (Fixpunkt-)Sätze

Warum Verifikation?

Are floating-point computations reliable?

Evaluate [Rump 2010]

$$f := 21 \cdot b \cdot b - 2 \cdot a \cdot a + 55 \cdot b \cdot b \cdot b \cdot b - 10 \cdot a \cdot a \cdot b \cdot b + a/(2 \cdot b)$$

with $a = 77617$ and $b = 33096$.

Single precision, double precision, and 80 bit arithmetic give same result:

$$\begin{aligned} f_{\text{float}} &\approx 1.172603\ldots \\ f_{\text{double}} &\approx 1.172603\ldots \\ f_{\text{80 bit}} &\approx 1.172603\ldots \end{aligned}$$

Is the first figure correct? Does $f > 0$ hold?

Mengenoperationen für $A, B \subset \mathbb{R}$:

$$A \circ B = \{a \circ b | a \in A, b \in B\} \qquad \text{unendlich viele Op.!}$$

Intervalloperationen für $A = [a_1, a_2], B = [b_1, b_2] \subset \mathbb{R}$:

$$A \circ B = [\ \min\{a_1 \circ b_1, a_1 \circ b_2, a_2 \circ b_1, a_2 \circ b_2\}, \max\{\ldots\}] \qquad \text{4 Op.!}$$

Bsp: $[-2, 1] * [4, 5] = [-10, 5]$

Intervallrechnung erlaubt die sichere numerische Berechnung von Einschließungen von Wertebereichen!

C++-Klassenbibliothek C-XSC kennt Datentyp `interval`

Value(s) of Rump's formula

```
template<typename T> T f(T x, T y) {
  // 21*y*y - 2*x*x + 55*y*y*y*y - 10*x*x*y*y + x/(2*y)
  // using arithmetic operations for type T operands ...
}
```

Main routine:

```
cout << "float result:  " << f( float(a), float(b) );
cout << "double result: " << f( double(a), double(b) );
cout << "interval result: " << f(interval(a), interval(b));
```

Output:

```
float result:      1.1726039...
double result:     1.1726039400531787
interval result:   [-8.190828E+003, 1.638518E+004]
```

Still: what about $f > 0$?

IEEE double precision is not enough

Interval arithmetic based on machine intervals with IEEE double numbers as bounds gives

$$\begin{aligned} f &= 21 \cdot b \cdot b - 2 \cdot a \cdot a + 55 \cdot b \cdot b \cdot b \cdot b - 10 \cdot a \cdot a \cdot b \cdot b + a/(2 \cdot b) \\ &\in \texttt{F} = \texttt{[-8.190828E+3, 1.638518E+4]} \end{aligned}$$

The fact that the interval `F` is wide even though the argument values $a = 77617$ and $b = 33096$ are machine-representable is a warning that roundoff and catastrophic cancellation have probably occurred.

Nevertheless, interval computations give much more information to the user than the traditional approach (computing in single and double precision).

Multiple-precision intervals in C-XSC

```
template<typename T> T f(T x, T y) {
  //formula as on previous slides ...
}
```

Staggered precision interval (data type `l_interval`) computation:

```
int a= 77617;
int b= 33096;
stagprec= 2; //use twofold double precision arithmetic
cout << "l_interval result: "
     << f( l_interval(a), l_interval(b) ) << endl;
```

Output:

```
l_interval result: [ -0.8273960600, -0.8273960599]
```

Result is now best possible

A second example: repeated squaring

```
template<typename T> T f(T x) {

  x= 1-x*x;

  for (int k=1; k<=80; k++) x= x*x;

  return x;   // (1-x*x)**(2^80)
}
```

Main routine:

```
  //s= 2^(-37) = 7.27596e-12
  cout << "f(s) using floats : " << f( float(s) );
  cout << "           doubles: " << f(double(s) );
  cout << "f(s) using ordinary intervals: "
       << f( interval(s) ) << endl;
```

Repeated squaring.
Again: IEEE double precision is not enough

The float and the double results are identical:

```
f(s) using floats : 1.00000
           doubles: 1.00000...
```

Nevertheless, the correct value is close to 1.6e-28

The ordinary interval computation produces a wide interval (warning!):

```
f(s) using ordinary intervals: [ 0.000000, 1.000000]
```

Using staggered intervals (stagprec=2):

```
f(s) using l_interval: [1.6038108905E-28,1.6038108906E-28]
```

Result again best possible

Verifizierte Nullstellensuche:
Intervall-Newton-Verfahren

Für $f \in C^1(\mathbb{R})$ definieren wir für Intervalle $X \subset \mathbb{R}$ und Punkte $x \in X$ den Intervall-Newton-Operator

$$N(X) := x - f(x)/f'(X).$$

Dabei bezeichnet $f'(X)^{-1}$ ein Intervall, das die Menge aller Ableitungen $\{f'(x)|x \in X\}$ umfasst.

Satz:
1.) Gilt $N(X) \subset X$, so hat f genau eine Nullstelle in X.
2.) Gilt $N(X) \cap X = \emptyset$, so liegt in X keine Nullstelle von f.

Beweis: klassischer mathematischer Beweis mittels Fixpunktsatz von Brouwer (Intervalle sind nichtleere, kompakte und konvexe Mengen).

Intervall-Newton-Verfahren

$$N(X) := x - f(x)/f'(X)$$

Intervallrechnung liefert Wertebereichseinschließungen für $f'(X)$ und $N(X)$.

$f'(X)$ wird mittels automatischer Differentiation eingeschlossen.

Aussage des Satzes, auf diese Einschließungen angewandt, ist hinreichend für die Gültigkeit des Satzes für die eingeschlossenen Größen selbst.

Es werden hinreichende Bedingungen vom Rechner automatisch verifiziert!

$N_{\text{computer}}(X) \subset X$ beweist also die Existenz einer Nullstelle.

$N_{\text{computer}}(X) \cap X = \emptyset$ beweist, dass in X keine Nullstelle von f liegt.

Insgesamt: mathematische Beweise mit Gleitkomma(intervall)rechnung.

Bemerkungen zur Verifikationsnumerik

Andere Bezeichnungen:

- Selbstverifizierende numerische Verfahren
- Numerik mit Ergebnisverifikation
- Self-validating/self-verifying numerical methods
- Reliable computing
- Numerical verification methods

Heutige Rechner: 10^{15} Gleitkommaoperationen pro Sekunde. Fehlerkontrolle muss automatisiert werden.

Warnung: Naive Intervallrechnung liefert meist unbefriedigende Ergebnisse!

Typisches Vorgehen: Berechne klassische numerische (Näherungs-)Lösung $\tilde{x}$. Schließe deren Defekt in Intervall X ein. Dann: exakte Lösung $\in \tilde{x} + X$

Proseminar zur Verifikationsnumerik im WS 2011/12

Typische Aufgabenstellung:

- Beschreibung der mathematischen Problemstellung (z.B. Lösung eines linearen Intervallgleichungssystems),
- Vorstellung eines bereits existierenden C-XSC-Programms zu dessen Lösung,
- Vorführung dieses Programms anhand einfacher konkreter Beispiele.

Interessent(inn)en sind herzlich willkommen!
Der Teilnahmeschein ist in verschiedenen Bachelor-POs verwertbar.

Vorlesung Verifikationsnumerik richtet sich typischerweise an Studierende im ersten Semester eines Masterstudiengangs.