Numerische Programmierung Konkrete Mathematik

Literatur

- Numerik für Informatiker (Huckle/Schneider) = Numerische Methoden
- Folien voriger Semester
- Herzberger: Wissenschaftliches Rechnen
- Opfer: Numerik für Anfänger
- Überhuber: Computer-Numerik
- Kahaner/Moler/Nash: Numerical Methods and Software
- Dahmen/Reusken: Numerik für Ingenieure und Naturwiss.

I. Warum Numerik?

1. Zu Computer Science gehört auch die Numerik, genauso wie Datenbanken, Software Engineering.
Ursprung der Informatik ist die Numerik!

Alles was mit Computer zu tun hat, gehört zu CS!
z.B. - Rechner-Arithmetik, Parallelrechner, usw.
- Computergraphik **(Flächen, Kurven)**
- Bildverarbeitung (Komprimierung, Filtern, Analyse) **: JPEG**
- Soundverarbeitung: MP3
- Information Retrieval, Prozessverwaltung

2. Wissenschaftliches Rechnen

Verbund von Naturwissenschaft, Mathematik, Numerik, Informatik zur Lösung wiss. Probleme:

z.B. Wettervorhersage, Raketensteuerung, Strömungs-Simulation (NASA, BMW, Siemens,…)

3. Numerik zur Lösung von Informatik-Problemen

z.B. - Warteschlangen, Betriebsmittelzuteilung, und Stochastische Automaten,
- Neuronale Netze und Fuzzy-Logik,
- Räuber-Beute-Modelle zur Ressourcenverteilung

Voraussetzungen aus der Informatik:

- Zahldarstellung
- Programmiersprache (JAVA)
- (Komplexitätstheorie)

Zur Mathematik:

Numerik als Fortführung der ‚Reinen Mathematik'
mit anderen Mitteln:

z.B. bestimme - $\int_a^b e^{-t^2}\,dt$,

- Gleichungssysteme, Nullstellen
- Grenzwerte, Eigenwerte (Resonanzen)
- Interpolation

kontinuierlich versus diskret!

Voraussetzungen aus der Mathematik:

Taylor-Entwicklung und Mittelwertsatz, Ableitungen, Summen und Reihen, trigonometrische Funktionen, Lineare Gleichungssysteme, Vektoren und Matrizen, Normen, Komplexe Zahlen.

Falls Fragen zu mathematischen Grundlagen auftauchen:
Bitte mich sofort darauf aufmerksam machen und nachfragen!

II. Rechnerarithmetik und Rundungsfehler

Motto:

„Die natürlichen Zahlen hat der liebe Gott gemacht, alles andere ist Menschenwerk.“

L. Kronecker

“Nobody is perfect”

Osgood zu Daphne (Jerry) in ‘Some like it hot’

Beispiel: Digitalisierung von Audio – Audacity, JPEG

Problem: Endlichkeit!
Die Menge **IR** der reellen Zahlen besteht aus unendlich vielen Zahlen mit unendlich vielen Stellen!

Jeder Computer ist endlich!

2.1. Definition:

Die endliche Menge **M der in einem Rechner** darstellbaren Zahlen heißt

Maschinenzahlen

Wir unterscheiden ganzzahlige Maschinenzahlen und Maschinenzahlen für reelle Zahlen (=Gleitpunktzahlen).

1.Problem: Abbildung **R → M liefert Fehler**

2.Problem: Arithmetische Operationen + - * / :
M x M → M gilt nicht!

Zum ersten Problem:

2.2 Definition:

Eine Abbildung rd: R → M bezeichnen wir

als Rundung, wenn gilt: $|x - rd(x)| = \min_{m \in M} |x - m|$

Dies ist keine eindeutige Definition von rd !

Andere Möglichkeiten statt rd: ceil, floor, to zero

Minimalforderung: $rd(m) = m \quad für \quad m \in M$

Rundungsfehler (absoluter): $f_{rd}(x) := x - rd(x)$

Zum zweiten Problem (arithmetische Operationen) :

$$+_M : \quad M \times M \mapsto M$$

$$x +_M y := rd(x + y)$$

Genauso definieren wir für - , * , /

die Näherungsoperatoren $-_M$, $*_M$, $/_M$

Beispiele von Maschinenzahlen:

Quantisierung von Musik- oder Bildwerten: 8 Bit, 16 Bit

Geldbeträge, Wechsel/Aktienkurse: DAX 3577,72

Alternativen zu Maschinenzahlen und -arithmetik?

- Rechnen mit beliebig vielen Stellen (symbolisches Rechnen, MATHEMATICA, MAPLE)
- Rechnen mit Intervallen (Intervall-Arithmetik)

Folien-Beispiel: 0.1114 + 0.001116

Modell: Jede Zahl repräsentiert durch drei Dezimalstellen und Position des Dezimalpunkts:

0.1114 → 0.111 und 0.001116 → 0.00112

Daher 0.111 + 0.00112 = 0.11212 → 0.112

Also Resultat: 0.112

Exaktes Ergebnis bei voller Stellenzahl:

0.1114 + 0.001116 = 0.112516

Volle Stellenzahl führt zu aufwendigen Rechnungen,

Zahlen würden zu lang!

Speicherplatz!

Rechenzeit!

Intervallarithmetik auf 3 Stellen (Folien-Beispiel):

Repräsentiere Zahlen durch Einschließungsintervalle:

$$0.1114 \in [0.111 \quad 0.112]$$

$$0.001116 \in [0.00111 \quad 0.00112]$$

Untere Intervallgrenze der Lösung:

$$\lfloor 0.111 + 0.00111 \rfloor = \lfloor 0.11211 \rfloor = 0.112$$

Obere Grenze:

$$\lceil 0.112 + 0.00112 \rceil = \lceil 0.11312 \rceil = 0.114$$

Resultat: Lösung von 0.1114 + 0.001116 liegt im Intervall [0.112 , 0.114]

Vorteil: Exakte Information über Lage und Qualität der Lösung

Nachteil: Rechenzeit! Ev. Ergebnisintervall sehr groß!

2.3. Definition: Festkommazahlen als Maschinenzahlen

$$n_1 n_2 \cdots n_k , m_1 \cdots m_j$$ mit

$$n_1, n_2, \ldots, n_k, m_1, \ldots, m_j \in \{0,1,\ldots,9\}, \quad k, j \in IN$$

Beispiele:

- Geldbeträge wie Euro.Cent , j=2
- Wechselkursangaben, wie 1.1591 ($ zu €), j=4, k=1
- Börsenindizes: DAX 3577.72, j=2, k=4.
- Alte Taschenrechner verwendeten Festpunktzahlen.

Für praktisches Rechnen ungenügend**!**

Zu wenige ganze, bzw. reelle Zahlen darstellbar!

Sehr große Zahlen? Sehr kleine Zahlen? Zahlen mit vielen Nachkommastellen? Zusätzlich große Rundungsfehler.

Börsenindex Vancouver 1983.

Start des Indexes mit Wert 1000.

Bei jedem Verkaufereignis (ca. 3000 pro Tag) wurde der Index neu berechnet auf drei Stellen nach dem Komma:

Rechne mit vier Stellen nach dem Komma

und dann Abschneiden der vierten Stelle.

(Quasi rechnen wie mit ganzen Zahlen in C)

Nach 22 Monaten wurde 574.081 angegeben.

Der ‚wahre' Wert: 1098.892

Systematischer Fehler, der sehr oft auftritt!

Landtagswahl 1992 in Schleswig-Holstein:

Grüne erhielten 4.97%.
Zur Darstellung der Ergebnisse Festkomma
mit einer Stelle nach dem Komma, (j=1);
also Anzeige des Ergebnisses als 5.0%.
Fehler wurde erst entdeckt, nachdem offizielle
Ergebnisse bereits veröffentlicht waren.

Rechnen(?) mit Exel!!!
Diese Zahlendarstellungen werden wir i.F. nicht verwenden

2.4. Integer(Maschinen)zahlen

Endlicher Ausschnitt aus den ganzen Zahlen **Z** symmetrisch um Null angeordnet.

Man verschiebt den Ausschnitt so, dass alle Zahlen positiv werden:

Menge der Integer-Maschinenzahlen M

$$z = \sum_{i=0}^{t-1} m_i \times 2^i - 2^{t-1}, \qquad m_i \in \{0,1\}, \quad i = 0,\ldots,t-1$$

$t \in IN$ **gibt die Stellenzahl an, bzw. Bits.**

Stufenzahlen: 2^i mit $0 \leq i \leq t-1$

Also $M = < -2^{t-1}, 2^{t-1}-1 >$

Bei 32 Bits (=4 Bytes) ergibt sich der Zahlenbereich der ganzzahligen Maschinenzahlen daher zwischen

$-2^{31} = -2147483648$ und $2^{31}-1=2147483647$

Folien-Beispiel:

Darstellung der Dezimalzahl 11 mit t = 5

$11 = 27 - 16$ *(da 11+16=27)*

$= (16 + 11) - 16 = 16 + (8+3) - 2^4$

$= 16 + 8 + (2+1) - 2^4$

$= 1*2^4+1*2^3+0*2^2+1*2^1+1*2^0 - 2^4$

→ $(11011)_2$, nicht $11=(01011)_2$

oder $-11 = 5 - 16$ → $(00101)_2$

Zu Beachten:

- Fehler bei Bereichsüberschreitung (Overflow)
  ev. *Wrap-around-Effekt.*
- Integer-Division in Programmiersprachen: 1 / 3 = 0
  Division ohne Rest oder Rundung zur Null
  (Abschneiden).
- Division durch 0 (*Beispiel: USS Yorktown)*

Vorteil: Null hat nur eine Darstellung!

Sonst treten bei Integerzahlen keine Rundungsfehler auf!

2.5. Gleitpunktzahlen

Stelle $x \in IR$ durch Vorzeichen, Mantisse m und Exponenten e dar, bzgl. Basis b>1:

$$x = (-1)^{v} \cdot m \cdot b^{e}$$

Wir betrachten nur b=2 (b=8,10,16 kommen kaum vor)

Beispiel:

Darstellungen der Zahl 16:

$$\cdots = 0.25 \cdot 2^{6} = 0.5 \cdot 2^{5} = 1 \cdot 2^{4} = 2 \cdot 2^{3} = 4 \cdot 2^{2} = \cdots$$

mit v=0, m = 2^{-i} und e = i+4 .

Normierung ist notwendig, damit Darstellung eindeutig:

Der Exponent ist stets so zu wählen, dass die Mantisse m genau eine von Null verschiedene Stelle <u>vor dem</u> Komma hat.

Also in unserem Beispiel $16 = +\ 1.0 \cdot 2^4$

Vorsicht: andere Bücher normieren so, dass erste Stelle <u>hinter dem Komma von Null verschieden!</u>

(also $16 = +\ 0.5 \cdot 2^5 = +\ (0.1)_2 \cdot 2^5$)

2.6. Definition: Normierte Gleitpunktzahlen

Die Menge M der (reellen) Maschinenzahlen besteht aus Zahlen der Form

$$x = (-1)^{\nu} \cdot \sum_{i=0}^{t-1} x_i 2^{-i} \cdot 2^{e}$$

Vorzeichen: $\nu \in \{0,1\},$

Mantisse: $x_0 = 1, \quad x_i \in \{0,1\} \quad sonst,$

Exponent: **Integerzahl e.**

Also hat die Mantisse eigentlich t Stellen;

aber die erste Stelle muss nicht gespeichert werden, da sie wegen der Normierung 1 ist.

Daher werden für die Mantisse nur (t-1)-Bits gebraucht!

Bei Zahlen nahe 0 kann die Normierung aufgehoben werden (sog. subnormale Gleitpunktzahlen)!

Ausnahmeregeln, falls Exponent minimal oder maximal ist!

Exponent wird gespeichert als ganzzahlige Integer-Maschinenzahl wie vorher beschrieben!

Folien-Beispiel:

13.6 = 8 + 5.6

= 8 + (4+1.6)

= 8 + 4 + (1+0.6)

= 8 + 4 + 1 + (1/2 + 0.1)

= 8+4+1+1/2 + (0.0625 + 0.0375)

= 8+4+1+1/2+1/16 + (0.03125+0.00625)

Also im Zweiersystem

$13.6 = (1101.10011...)_2$

Als normierte Gleitpunktzahl:

$$13.6 = (-1)^0 \cdot (1.10110011\ldots)_2 \cdot 2^3$$ mit e=3.

Die Zahl 13.6 in verschiedenen Genauigkeiten:

$$(1.10110011\ldots)_2 * 2^3$$

t	*Darstellung*	*Rundung*	*Fehler*
2	$(1.1)_2 * 2^3$	*ab*	$(1.6)_{10}$
3	$(1.11)_2 * 2^3$	*auf*	$(0.4)_{10}$
4	$(1.110)_2 * 2^3$	*auf*	$(0.4)_{10}$
5	$(1.1011)_2 * 2^3$	*ab*	$(0.1)_{10}$
6	$(1.10110)_2 * 2^3$	*ab*	$(0.1)_{10}$
7	$(1.101101)_2 * 2^3$	*auf*	$-(0.025)_{10}$

Anzahl der für Mantisse und Exponent benutzten Bit und daraus sich ergebende Rechengenauigkeit und

Exponentenbereich bei IEEE-Datentyp float und double:

Typ	*Mantisse*	*t*	*Exponent*	$[e_{\min}, e_{\max}]$	ε
float	$23(+1)$	8	$[-126, 127]$	$2^{-24} \approx 6*10^{-8}$	
double	$52(+1)$	11	$[-1022, 1023]$	$2^{-53} \approx 1*10^{-16}$	

IEEE-Standard (single precision, 32 Bit):

Exponent e mit 8 Bit, gespeichert in der ‚positiven' Form

$$p = e + 127$$

$p=(0000\ 0000)_2$ ist dann kleinstmöglicher Exponent.
Ist auch noch in der Mantisse (bis auf Normierung)
alles 0, so wird diese Zahl als Null interpretiert.

Entsprechend $p=(1111\ 1111)_2 = 255$
als unendlich (NaN oder Not a number)

$-126 <= e <= 127$ entspricht $1 <= p <= 254$

Für die Mantisse bleiben 23 Bit
(24 unter Berücksichtigung der Normierung).
1 Bit für Vorzeichen → Insgesamt 1+23+8=32 Bit .

Beispiel Ariane 5

1996 endete die erste Ariane 5 durch Selbstzerstörung

40 sec nach Start.

Ursache:

36.7 sec nach Start versuchte der Bordcomputer den Wert der horizontalen Geschwindigkeit von 64 Bit Gleitpunkt in 16 Bit signed Integer umzurechnen.

Der sich ergebende Wert war zu groß →

→ Overflow

→Absturz des Computers,

→Übergabe an Back-up-Rechner, der aber aus demselben Grund bereits abgestürzt war

→Kein Lenksystem mehr

→instabiler Flug

→ Selbstzerstörung.

Benutzte Software stammte von Ariane 4.

Ariane 5 war schneller!

Umwandlung war nicht abgesichert!

Bereits Konrad Zuse verwendete in seinen Z1-Z4 Gleitpunktzahlen.

Rundung

2.7. Def.: Für $x = (-1)^v \cdot 2^e \cdot 1.x_1x_2...x_{t-1}x_t...$ definieren wir

$$rd(x) = (-1)^v \cdot 2^e \cdot 1.x_1x_2\ldots x_{t-1} \quad \textit{für} \quad x_t = 0$$

$$rd(x) = (-1)^v \cdot 2^e \cdot 1.x_1x_2\ldots x_{t-1} + 2^{-t+1} \quad \textit{für} \quad x_t = 1 \quad \textit{und} \quad x_tx_{t+1}x_{t+2}... \neq 1000..$$

$$rd(x) = (-1)^v \cdot 2^e \cdot 1.x_1x_2\ldots x_{t-1} \quad \textit{für} \quad x_t = 1 \quad \textit{und} \quad x_{t-1}x_tx_{t+1}... = 0100..$$

$$rd(x) = (-1)^v \cdot 2^e \cdot 1.x_1x_2\ldots x_{t-1} + 2^{-t+1} \quad \textit{für} \quad x_t = 1 \quad \textit{und} \quad x_{t-1}x_tx_{t+1}... = 1100..$$

1. Fall: Abschneiden, falls $x_t = 0$;

2. Fall: Letztes Bit wird um eins erhöht, falls

x_t=1 und $x_tx_{t+1}x_{t+2}\ldots \neq 1000\ldots$

3. Fall: (x_t=1) $rd(1.x_1x_2\ldots x_{t-2}0|1) = 1.x_1x_2\ldots x_{t-2}0$

oder $rd(1.x_1x_2\ldots x_{t-2}1|1) = 1.x_1x_2\ldots x_{t-2}1 + 2^{-t+1}$

Rundung so, dass letztes Bit 0 wird.

Es kann ev. Overflow auftreten (zu großer Exponent).

Dieser Fall wird aber im Folgenden ignoriert (Fehlermeldung?)

Es kann Underflow auftreten.

In der Regel wird dann einfach die kleine Zahl zu 0 gesetzt.

Die Rundungsfehleranalyse in den folgenden Abschnitten wird nur für Normalfall durchgeführt (ohne Over/Underflow)

Dabei ist nur die Mantisse wichtig!

Der Exponent spielt keine Rolle, weil dabei keine Fehler auftreten.

2.8. Absoluter Rundungsfehler:

$$f_{rd}(x) = x - rd(x)$$

$$\left| f_{rd}(x) \right| = \left| x - rd(x) \right| \leq \frac{1}{2} 2^{-t+1} 2^{e} = 2^{e-t}$$

Problem: Ein absoluter Fehler von der Größe 0.1 ist
- bei der Zahl 2.1 recht groß, aber
- bei der Zahl 123456.7 sehr klein.

Beispiel: 1 Million + 1 Jahr alter Dinosaurierknochen

Daher sinnvollere Definition durch

2.9. Relativer Rundungsfehler:

$$f_{rel}(x) := \varepsilon_x := \frac{f_{rd}(x)}{x} = \frac{\delta_x}{x} = \frac{x - rd(x)}{x} \quad \textbf{für} \quad x \neq 0$$

Dann gilt: $$\left| f_{rel}(x) \right| \leq \frac{2^{e-t}}{|x|} \leq \frac{2^{e-t}}{2^e} = 2^{-t}$$

da wegen Normierung die Mantisse in [1,2] liegt:
Mantisse stets $(1.bbbbb...)_2 < 2$
also m >= 1, d.h. |x| >= 2^e

Außerdem gilt durch Umformung von 2.9:

2.10.: $$rd(x) = x - x\varepsilon_x = x(1 - \varepsilon_x)$$

mit $$|\varepsilon_x| \leq 2^{-t}$$

Gerundete Zahl=Ausgangszahl, bis auf Faktor (1±ε)

Def.: Die obere Schranke für den relativen Fehler, der bei der Rundung mit t-stelliger Mantisse auftreten kann, heißt **Maschinengenauigkeit ε,** und ergibt sich als

$$\varepsilon = 2^{-t}$$

Andere Möglichkeit, die Maschinengenaugkeit zu definieren:
Größte positive Zahl $y=2^{-k}$, so dass

$$1.0 + y = 1.0$$

Beispiel t=2, $\varepsilon = ¼ = (0.01)_2$; $(1.0|1)_2 \rightarrow (1.0)$

Mantissenlänge (Bits) ←→ Genauigkeit

Gleitpunktarithmetik

2.12. Def. (Realisierung einer Maschinenoperation):

- Berechne für Maschinenzahlen das Ergebnis der Operation mit höherer Genauigkeit (quasi exakt)

- Runde dieses Resultat wieder auf Maschinenzahl.

Dadurch ist der auftretende Fehler ausschließlich gegeben durch den Rundungsfehler, der im letzten Schritt auftritt!

Beispiel Addition $+_M$:

Ausgangspunkt: Normierte Gleitpunktdarstellung beider Zahlen

- Verschiebe bei einer Zahl den Exponenten, so dass beide Zahlen den gleichen Exponenten haben.
- Addiere nun die Mantissen.
- Normalisiere das Ergebnis (verschiebe das Komma).
- Runde das Ergebnis.

Folien-Beispiel: x=7/4 und y=3/8 → x+y=17/8

Mantisse mit t=3

$(1.11)_2\ 2^0 +_M (1.10)_2\ 2^{-2} =$

$= (111)_2\ 2^{-2} + (1.10)_2\ 2^{-2}$

$= (1000.10)_2\ 2^{-2}$

$= (1.00010)_2\ 2^1$

$= (1.00)_2\ 2^1$.

Also x + y = 17 / 8 , aber $x +_M y = 2$.

Absoluter Fehler : |17/8 – 2| = 1/8

Relativer Fehler: $\left|\frac{1/8}{17/8}\right| = 0.0588\ldots \cong 6\%$

Zum Vergleich: Bei t=3 ist die Maschinengenauigkeit

$$\varepsilon = 2^{-3} \cong 12.5\%$$

Der auftretende Fehler ε_+ der Gleitpunktaddition entsteht also durch die abschließenden Rundung!

Nach 2.10 und 2.11 gilt daher

2.13. $$x +_M y = rd(x + y) = (x + y)(1 + \varepsilon_+)$$

mit $|\varepsilon_+| \leq \varepsilon$ Maschinengenauigkeit

In der Praxis ersetzt man die exakte Addition der Mantissen (Schritt 2) durch eine Addition mit höherer Genauigkeit, meist mit doppelter Genauigkeit. Danach Rundung auf Maschinenzahl.

Ähnliches Modell bei Multiplikation / Division und auch bei anderen Funktionsauswertungen.

Beispiel: Realisierung der Gleitpunkt-Division in INTEL-Prozessor und INTEL-Pentium-Bug 1994.