Mathematik und Programmierung für BI-M

Dr. Patrick Kürschner
patrick.kuerschner@htwk-leipzig.de
Z-415
Mathematisch-Naturwissenschaftliches Zentrum

12.10.2020

HTWK
Hochschule für Technik,
Wirtschaft und Kultur Leipzig

Planung und Organisation

Planung und Organisation

- OPAL-Plattform

  `https://bildungsportal.sachsen.de/opal/auth/RepositoryEntry/9631432726/CourseNode/87893545727080`

- Termine
  - Vorlesung (1 SWS) online: Montag, 11:15-12:45 Uhr, 14-tägig
  - Seminare ($\frac{1}{2}$ SWS) online in 4 Gruppen [GSW, BB, HBB, KI], 4 Termine (KW 43, 47, 51, 55)
  - Computer-Praktika ($\frac{1}{2}$ SWS) in 7 Gruppen [GSWa/b, BB, HBBa/b, KIa/b], Präsenz in G127-L, G129-L, 3 Termine (KW 45, 49, 57)
    HTWK-Hygienekonzept + Regelung F Bau zur Nutzung der PC-Pools beachten:
    - Dokumentation Anwesenheit erforderlich
    - Mund-Nase-Schutz bei Betreten/Verlassen

Planung und Organisation

- Lernziele
  - Grundlegendes Verständnis der Verfahrensweise numerischer Methoden für Standardprobleme
  - Fähigkeit diese anzuwenden und zu beurteilen, d.h. kritische Analyse von Berechnungsergebnissen
  - Allgemeine Fähigkeit für gegebenes Problem passende Verfahren auszuwählen
- Zulassung zur Klausur: Abgabe Online-Belegaufgaben, 4-5 während Semester, inkl. Programmieranteile
- Klausur: 90 min.

Übersicht

Vorkenntnisse: Das Modul richtet sich an Master-Studierende des 1. Semesters.

→ Grundlegende Mathematikkenntnisse (Mathe I + II) aus dem Bachelorstudium können vorausgesetzt werden.

Insbesondere werden im Rahmen der LV Kenntnisse benötigt zu:

- Lineare Algebra (Vektoren, Matrizen, lineare Gleichungssysteme),
- Differentialrechnung in einer und mehrerer Veränderlichen ((partielle) Ableitungen, Taylorentwicklung, Gradient, Jacobi-Matrix).

Aufgrund des Geringen Stundenumfangs (2 SWS) können diese Grundlagen nicht nochmal detailliert wiederholt werden ⇒ ein eventuell notwendiges Wiederauffrischen ist im Selbststudium zu erledigen.

→ Ebenfalls empfohlen&wünschenswert: Grundlagen Programmierung

Übersicht

Anonymer Fragebogen über Vorkenntnisse:

	Noch bekannt	Schon mal gehört, vage in Erinnerung	unbekannt, vergessen
Lineare Gleichungsysteme $Ax = b$ und Gauß'sches Lösungsverfahren	57,14 %	32,14 %	7,14 %
Kleinste-Quadrate-Methode	7,14 %	17,86 %	67,86 %
Lineare Regression	21,43 %	46,43 %	32,14 %
Newton Verfahren	21,43 %	42,86 %	32,14 %
Partielle Ableitungen	39,29 %	46,43 %	14,29 %
Gradient	14,29 %	67,86 %	14,29 %
Jacobi-Matrix	3,57 %	21,43 %	71,43 %
Taylor-Reihenentwicklung	10,71 %	50 %	35,71 %

sehr ähnlich

wird später gezeigt

Übersicht

Anonymer Fragebogen über Vorkenntnisse:

Mit welchen Programmiersprachen bzw. mit welchen mathematischen und numerischen Softwa

26,32 %	☐	MATLAB / OCTAVE
0 %	☐	Julia
0 %	☐	Python (numpy, Scipy, tensorflow, ...)
63,16 %	☐	Computeralgebrasysteme (Maple, Mathematica, graphische Taschenrechner, ...)
21,05 %	☐	(Höhere) Programmiersprachen wie, z.B., C/C++, Fortran, Pascal, ...

Übersicht

Inhalte:

1. Motivation Numerisches Rechnen
2. Einführung in Softwareumgebung MATLAB® ($\rightarrow$ Praktika)
3. Maschinenzahlen und Phänomene des Rechnens in endlicher Arithmetik
4. Numerische Lösung linearer und nichtlinearer Gleichungssysteme
5. Interpolation und Approximation

Softwareumgebung

MATLAB: Software von Mathworks zum numerischen Rechnen, weit verbreitet in Wissenschaft, Industrie und Wirtschaft

Alle HTWK Studierenden & Angestellten erhalten eine kostenlose Lizenz für MATLAB:

---> siehe Hinweise im OPAL

`https://itsz.htwk-leipzig.de/serviceangebote-dienste/software/matlab/`

Nichtkommerzielle Alternative zu MATLAB: GNU Octave (weitgehend identischer Syntax).

Es wird wärmstens empfohlen MATLAB oder GNU Octave auf dem eigenen PC / Laptop zu installieren. Denn:

- Nur sehr geringe Stundenzahl im PC-pool
- für Selbststudium wertvoll
- MATLAB-Demonstrationen in Vorlesung (Codes werden bereitgestellt)
- Programmieranteile in Seminaren / Belegen.

Einführung und Motivation Numerische Mathematik

Einführung und Motivation Numerische Mathematik

Numerische Mathematik:

- Entwicklung und Analyse von Verfahren/Algorithmen zur näherungsweisen Lösung mathematischer Probleme.
  - Es geht wirklich um das konkrete Berechnen einer Lösung.
  - Nachweis Existenz (& Eindeutigkeit) einer Lösung nicht genug.
- Schnittstelle zwischen Theorie (Analysis, Lineare Algebra, ...) und realem Anwendungsbereich (Bauingenieurswesen, Maschinenbau, E-Technik, Physik, Biologie, Wirtschaft, ...)

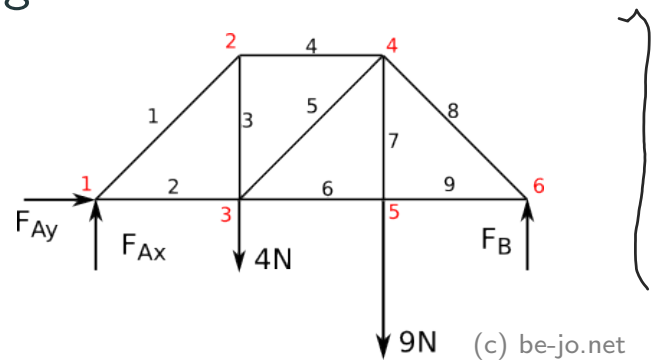
Einführung und Motivation Numerische Mathematik

Der Lange Weg vom Problem bis zur Lösung:

1. (Mathematische) Modellierung. Übersetzung des vorliegenden Problems in die Sprache der Mathematik: (Bewegungs)Gleichungen, Formeln, ...
2. Modellanalyse: theoretische Untersuchung des Modells auf Existenz und Eindeutigkeit einer Lösung (Analysis, Algebra, ...).
3. Entwicklung numerischer Lösungs- oder Approximationsverfahren notwendig, weil
   - dass Modell nicht exakt gelöst werden kann, z. B. Lösung von $e^x - 1 = x$
   - die Rechnung „von Hand“ kommt nicht in Frage, z. B. Lösen eines Gleichungssystems mit 100 Gleichungen und 100 Unbekannten. $Ax = b$
4. Implementierung von (effizienten) dem Problem angepassten Algorithmen am Rechner
5. Analyse der Ergebnisse. Plausibilitätscheck und Fehlerrechnung durchführen, weil der Rechner
   - die meisten Zahlen nicht exakt darstellen kann und Rechnungen nicht exakt ausführen kann

Beispiel aus Mechanik: Berechnung von Kräften im Fachwerk

1. Reales Problem und Modellierung



$\rightarrow$ lineares Gleichungssystem $Ax = b,\ A \in \mathbb{R}^{n\times n},\ b \in \mathbb{R}^n$

2. Modellanalyse: (eindeutige) Lösbarkeit, d.h. $\det(A) \neq 0$?
3. Entwicklung numerisches Lösungsverfahren: z. B. Gauß-Algorithmus, ...
4. Implementierung: z.B. in MATLAB, GNU Octave, ...
   Anwendung implementierter Algorithmus auf $Ax = b$
   $\rightarrow$ numerisch berechnete Näherungslösung $\tilde{x}$
5. Analyse: $\tilde{x} \approx x$?

Zahlendarstellung im Computer

Zahlendarstellung im Computer

Bereits bekannt:

Dezimalzahlen

Darstellung reelle Zahl $x \in \mathbb{R}$ in Basis 10:

$$x = \pm[\underbrace{z_n z_{n-1} \dots z_1 z_0}_{\text{Vorkommastellen}} . \underbrace{z_{-1} \dots z_{-k} \dots}_{\text{Nachkommastellen}}]_{10}$$

$$= \pm(z_n 10^n + z_{n-1} 10^{n-1} + \dots + z_1 10^1 + z_0 10^0 + z_{-1} 10^{-1} + \dots + z_{-k} 10^{-k} + \dots)$$

mit Vorzeichen ± 1 und Ziffern $0 \leq z_i \leq 9$.

Beispiel:

- $[-242.625]_{10} =$
  $-\left(2 \cdot 10^2 + 4 \cdot 10^1 + 2 \cdot 10^0 + 6 \cdot 10^{-1} + 2 \cdot 10^{-2} + 5 \cdot 10^{-3}\right)$
- $[\frac{1}{3}]_{10} = 0.\overline{3} = 0.3333\ldots = 3 \cdot 10^{-1} + 3 \cdot 10^{-2} + \dots$

Zahlendarstellung im Computer

Der Computer arbeitet intern hingegen mit Bits (0,1) als kleinste Speichereinheit.

Binärzahlen

Darstellung reelle Zahl $x \in \mathbb{R}$ in Basis 2:

$$x = \pm[\underbrace{z_n z_{n-1} \dots z_1 z_0}_{\text{Vorkommastellen}} . \underbrace{z_{-1} \dots z_{-k} \dots}_{\text{Nachkommastellen}}]_2$$

$$= (-1)^s \cdot (z_n 2^n + z_{n-1} 2^{n-1} + \dots + z_1 2^1 + z_0 2^0 + z_{-1} 2^{-1} + \dots + z_{-k} 2^{-k} + \dots)$$

mit Vorzeichen**bit** $s \in \{0, 1\}$ und Ziffern/**Bits** $z_i \in \{0, 1\}$.

Beispiele:

- $[-242.625]_{10} =$
- $[\frac{1}{3}]_{10} =$

Zahlendarstellung im Computer

Beispiel Umrechnung dezimal zu binär:

$[-242.625]_{10} = -[11110010.101]_2$

Vorzeichen: $-$

Vorkommastellen
wiederholtes ganzzahliges Teilen durch 2 mit Rest bis Quotient=0

	Quotient	Rest
242:2=	121	0
121:2	60	1
60:2	30	0
30:2	15	0
15:2	7	1
7:2	3	1
3:2	1	1
1:2	0	1

Nachkommastellen
wiederholtes Multiplizieren mit 2 bis Ergebnis=1

	Ergebnis	≥ 1?
$2 \cdot 0.625 =$	1.25	1
$2 \cdot 0.25$	0.5	0
$2 \cdot 0.5$	1	1

Zahlendarstellung im Computer

Der Computer arbeitet intern hingegen mit Bits (0,1) als kleinste Speichereinheit.

Binärzahlen

Darstellung reelle Zahl $x \in \mathbb{R}$ in Basis 2:

$$x = \pm[\underbrace{z_n z_{n-1} \dots z_1 z_0}_{\text{Vorkommastellen}} . \underbrace{z_{-1} \dots z_{-k} \dots}_{\text{Nachkommastellen}}]_2$$

$$= (-1)^s \cdot (z_n 2^n + z_{n-1} 2^{n-1} + \dots + z_1 2^1 + z_0 2^0 + z_{-1} 2^{-1} + \dots + z_{-k} 2^{-k} + \dots)$$

mit Vorzeichen**bit** $s \in \{0, 1\}$ und Ziffern/**Bits** $z_i \in \{0, 1\}$.

Beispiele:

- $[-242.625]_{10} = -(128 + 64 + 32 + 16 + 2 + \frac{1}{2} + \frac{1}{8}) =$
  $-(1 \cdot 2^7 + 1 \cdot 2^6 + 1 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1$
  $+0 \cdot 2^0 + 1 \cdot 2^{-1} + 1 \cdot 2^{-3}) = -[11110010.101]_2$
- $[\frac{1}{3}]_{10} = \dots = [0.010101\dots]_2 = [0.0\overline{10}]_2$
  $\rightsquigarrow$ unendlich viele Bits nötig, Computer hat aber nur endlichen Speicher!

Zahlendarstellung im Computer

Binäre Gleitpunktzahlen mit m Stellen (engl: floating point numbers)

Schreibweise

$$x = [s.z_1 z_2 \dots z_m]_2 \cdot 2^E = (-1)^s \cdot (z_1 2^{-1} + \dots + z_m 2^{-m}) \cdot 2^E$$

wobei

- Vorzeichenbit $s \in \{0, 1\}$, Ziffern/Bits $z_i \in \{0, 1\}$.
- ganzzahliger, vorzeichenbehafteter Exponent $E \in \mathbb{Z}$ (binär codiert)
- $[.z_1 z_2 \dots z_m]_2$ ist die **Mantisse** mit $m < \infty$ Stellen

Normierte Gleitpunktdarstellung für $x \neq 0$: Mantisse $[.z_1 z_2 \dots z_m]_2$ mit $z_1 \neq 0$ durch Anpassung des Exponenten (sog. Punktverschiebung)

Beispiel:

- $[-242.625]_{10} = -[11110010.101]_2$
  $\rightsquigarrow [1.1110010101]_2 \cdot 2^{[8]_{10}} = [1.1110010101]_2 \cdot 2^{[1000]_2}$

Zahlendarstellung im Computer

Wieviel Speicher (Bits) pro Gleitpunktzahl?

Die IEEE (Institute of Electrical and Electronics Engineers) - Norm 754
gibt seit 1985 zwei Typen von Gleitpunktzahlen (Maschinenzahlen) vor:

Typ	**Speicher**	**Vorzeichen**	**Mantisse**	**Exponent**
single precision	32 Bit	1 Bit	23 Bit	8 Bit, $E \in [-126, 127]$
double precision	64 Bit		52 Bit	11 Bit, $E \in [-1022, 1023]$

- Zahlen, die nicht in das Schema passen, führen zu Fehlern!
- Elimination des Vorzeichens von E durch "Trick": speichere *verschobenen Exponenten* $E_B = E + B$ mit Bias/Versatz B

$$x = [s.z_1 z_2 \dots z_m]_2 \cdot 2^{E_B - B},$$

  wobei $B = 127$ in single, $B = 1022$ in double precision
- *Maschinennull* codiert als $E_B = 000000 \dots 0$
- Bitmuster-Schreibweise $x =$ 

Zahlendarstellung im Computer

Beispiel von vorhin in single precision mit Bias $B = 127$, $m = 23$

$[-242.625]_{10} = [1.11110010101]_2 \cdot 2^8$

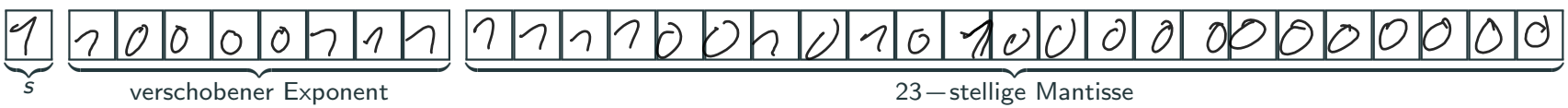
Vorzeichenbit: $s = 1$,

Mantisse: .11110010100...0

Exponent: 8

$\Rightarrow$ verschobener Exponent $E_B = E + B =$ 135

E_B Binär: $135 = 128 + 4 + 2 + 1 = 2^7 + 2^2 + 2^1 + 2^0 = [10000111]_2$



Zahlendarstellung im Computer

Die Menge $\mathbb{M}$ der verfügbaren Maschinenzahlen im Gleitpunktformat (single/double precision) ist offenbar diskrete und endliche Teilmenge der reellen Zahlen ($\mathbb{M} \subset \mathbb{R}$).

Rundung und Rundungsfehler

Zahl $x \in \mathbb{R}$ mit $x \notin \mathbb{M}$ muss zur nächstliegenden Gleitpunktzahl $\tilde{x} \in \mathbb{M}$ **gerundet** werden:

$$|x - \tilde{x}| \leq \min_{y \in \mathbb{M}} |x - y|$$

Es gilt für den **relativen Rundungsfehler**

$$\frac{|x - \tilde{x}|}{|x|} \leq 2^{-m+1} =: \text{eps} \quad \text{(Maschinenepsilon)}$$

sowie

$$\tilde{x} = x(1 + \epsilon), \quad |\epsilon| \leq \text{eps} \approx \begin{cases} 10^{-8} & \text{in single precision} \\ 10^{-16} & \text{in double precision} \end{cases}$$

Zahlendarstellung im Computer

Beim Arbeiten mit Maschinenzahlen in endlicher Genauigkeit ist Vorsicht geboten!

Katastrophales Beispiel:
Start Ariane 5 Rakete am 4. Juni 1996
Bordcomputer versuchte 64 Bit Gleitpunktzahl in 16 Bit Ganzzahl umzuwandeln.

- $\rightsquigarrow$ Fehler
- $\rightsquigarrow$ Bordcomputer stürzte ab
- $\rightsquigarrow$ Lenksystem versagte
- $\rightsquigarrow$ Absturz/Explosion

Gleitpunktarithmetik

Gleitpunktarithmetik

Bekannt: Seien $x, y \in \mathbb{R}$ und $\circ \in \{+, -, \cdot, \backslash\}$ eine arithmetische Operation

$\Rightarrow x \circ y \in \mathbb{R}$ (sofern definiert: $y \neq 0$ für Division)

Gilt nicht mehr für Maschinenzahlen:

Für $x, y \in \mathbb{M}$ ist im Allgemeinen $z = x \circ y$ keine gültige Maschinenzahl ($z \notin \mathbb{M}$)!

Beispiel: zur Einfachheit & Übersichtlichkeit mit 4-stelligen dezimalen Gleitpunktzahlen:

$$x = 43210 = 0.43210 \cdot 10^5, \quad y = 23.1 = 0.231 \cdot 10^2$$

$$z = x + y = 43233.1 = 0.432331 \cdot 10^5 \quad \text{6-stellig}$$

$\rightsquigarrow$ Müssen Ergebnis z zu 4-stelliger Maschinenzahl

$$\tilde{z} = 43230 = 0.4323 \cdot 10^5 \textbf{runden}!$$

Gleitpunktarithmetik

Gleitpunktoperation (floating point operation, flop)

Für Gleitpunktzahlen $x, y \in \mathbb{M}$ und eine arithmetische Operation $\circ \in \{+, -, \cdot, \backslash\}$ definieren wir eine arithmetische Gleitpunktoperation $\tilde{\circ}$ wiefolgt:

$$x \tilde{\circ} y = \mathrm{rd}(x \circ y) \in \mathbb{M} \quad \text{mit Rundungsoperation} \quad \mathrm{rd}(\ )$$

Bemerkung: die Operation $\circ$ wird dabei in etwas höherer Genauigkeit (z. B. mit Mantissenlänge $m + 1$) ausgeführt.

Standardmodell der Gleitpunktarithmetik

Für die Ergebnisse $\tilde{z}$ bzgl. Gleitpunktoperation gelte

$$\tilde{z} = x \tilde{\circ} y = \underbrace{(x \circ y)}_{=z,\text{exakt}}(1 + \delta), \quad |\delta| \leq \mathtt{eps} \quad \forall \circ \in \{+, -, \cdot, \backslash\}$$

Erfüllt, z. B., in IEEE single & double precision.

Gleitpunktarithmetik

Phänomene beim Rechnen in Gleitpunktarithmetik (d.h. in endlicher Genauigkeit):

- Assoziativ- und Distributivgesetz gelten im Allg. nicht für Gleitpunktoperationen.

  Beispiel: 3-stellige dezimale Gleitpunktzahlen $a = 2590$, $b = c = 4$:
  Exakt: $a + b + c = (2590 + 4) + 4 = 2590 + (4 + 4) = 2598$

  - $(2590 \tilde{+} 4) \tilde{+} 4$: $(2590 \tilde{+} 4) = 2594 \overset{\text{runden}}{\rightsquigarrow} 2590$, $(2590 \tilde{+} 4) \overset{\text{runden}}{\rightsquigarrow} 2590$
  - $(4 \tilde{+} 4) \tilde{+} 2590$: $4 \tilde{+} 4 = 8$, $8 \tilde{+} 2590 = 2598 \overset{\text{runden}}{\rightsquigarrow} 2600$

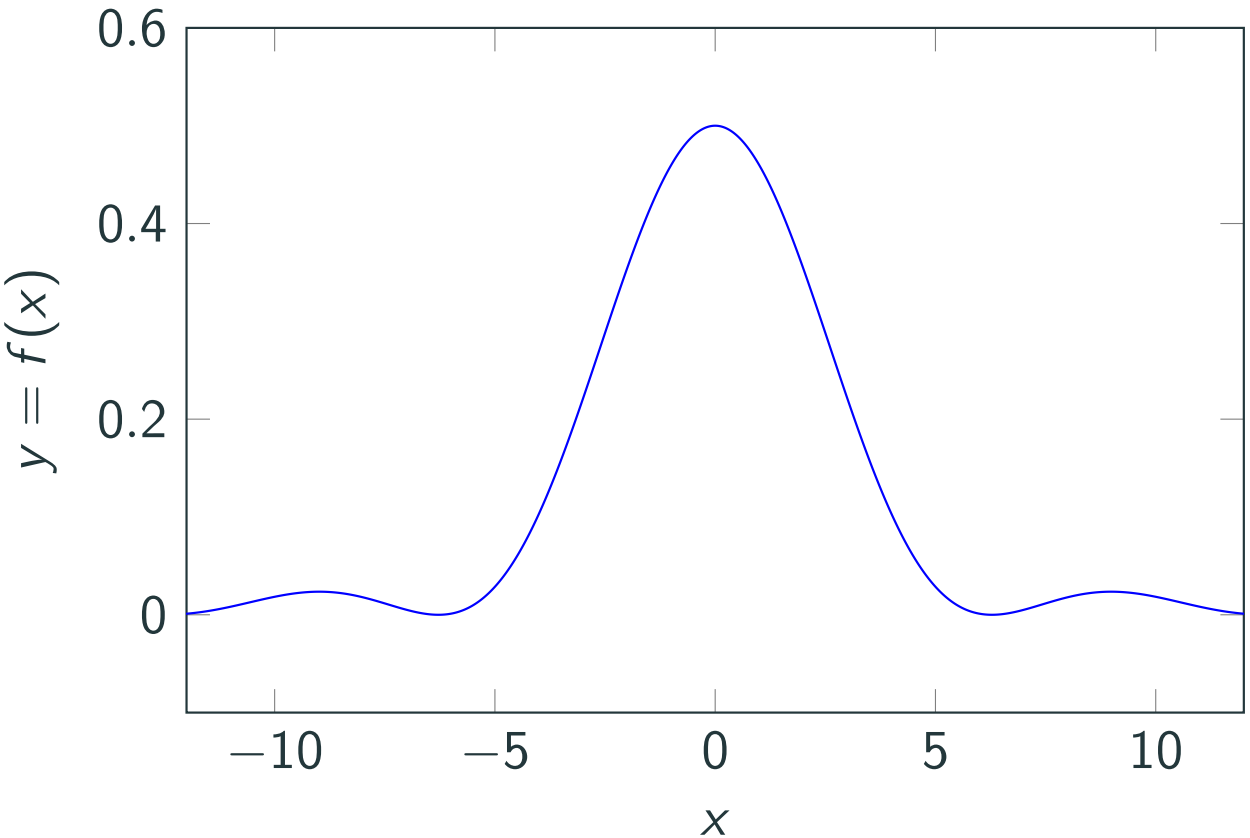
- Subtraktion $x \tilde{-} y$ mit $x \approx y$ kann zu großen Fehlern führen (Verlust signifikanter Stellen in Mantisse) „**Auslöschung**“
- Division $x \tilde{\backslash} y$ mit $y \approx 0$ ebenfalls

Gleitpunktarithmetik

Beispiel Auslöschung:

Betrachten

$$y = f(x) = \frac{1 - \cos(x)}{x^2} \quad \text{mit} \quad \lim_{x \to 0} f(x) = \frac{1}{2} \quad \text{(Übung)}$$



Gleitpunktarithmetik

Beispiel Auslöschung:

Betrachten

$$y = f(x) = \frac{1 - \cos(x)}{x^2} \quad \text{mit} \quad \lim_{x \to 0} f(x) = \frac{1}{2} \quad (\text{Übung})$$

Berechnen $f(x)$ für $x = 10^{-k}$, $k = 1, 2, \ldots, 8$ in *double precision* mit MATLAB.

Ergebnisse (ähnlich mit GNU Octave, Taschenrechner, ...):

k	$f(10^{-k})$
1	0.499583472197429
2	0.499995833347366
3	0.499999958325503
4	0.499999996961265
5	0.500000041370186
6	0.500044450291171
7	0.499600361081320
8	0

$\approx \frac{1}{2}$

Gleitpunktarithmetik

Beispiel Auslöschung:

Betrachten

$$y = f(x) = \underbrace{\frac{1 - \cos(x)}{x^2}} \quad \text{mit} \quad \lim_{x \to 0} f(x) = \frac{1}{2} \quad \text{(Übung)}$$

Berechnen $f(x)$ für $x = 10^{-k}$, $k = 1, 2, \ldots, 8$ in *double precision* mit MATLAB.

Ergebnisse (ähnlich mit GNU Octave, Taschenrechner, ...):

k	$f(10^{-k})$
1	0.499583472197429
2	0.499995833347366
3	0.499999958325503
4	0.499999996961265
5	0.500000041370186
6	0.500044450291171
7	0.499600361081320
8	0

Ausweg hier: analytische Umformung zu

$$\underbrace{g(x) = \frac{1}{2}\left(\frac{\sin\left(\frac{x}{2}\right)}{\frac{x}{2}}\right)^2 \equiv f(x)}$$

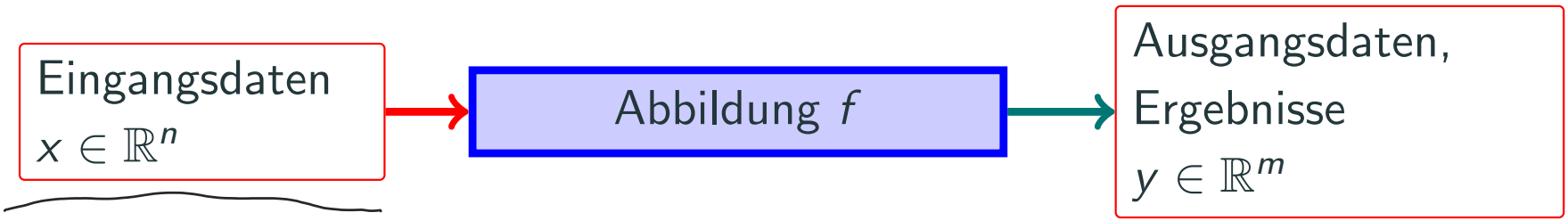
Verifikation als Übung

Fehlerfortpflanzung und Kondition

Fehlerfortpflanzung und Kondition

Abstraktes Modell für Berechnungen/Algorithmen:

Aus Eingangsdaten $x = (x_1, \ldots, x_n)^T \in \mathcal{D} \subset \mathbb{R}^n$ werden durch Folge arithmetischer Operation zu Ausgangsdaten/Ergebnissen $y = (y_1, \ldots, y_m) \in \mathbb{R}^T$ berechnet gemäß Abbildung $f : x \mapsto y$.



Beispiel: Lösung lineares Gleichungssystem $Ay = b$.
Daten A, b, Ergebnis $f : \{A, b\} \mapsto y = A^{-1}b$

Betrachten im Folgenden Abbildung als Funktion:

$$y = f(x) = \begin{bmatrix} f_1(x_1, \ldots, x_n) \\ \vdots \\ f_m(x_1, \ldots, x_n) \end{bmatrix}, \quad f : \mathcal{D} \subset \mathbb{R}^n \mapsto \mathbb{R}^m$$

Fehlerfortpflanzung und Kondition

Exakte Eingangsdaten x, Störung Δx

Gestörte Eingangsdaten $\tilde{x} = x + \Delta x$ (z. B. durch Rundung zu Gleitpunktzahlen)

$$\Rightarrow \text{gestörte Ausgangsdaten } \tilde{y} = y + \Delta y = f(\tilde{x}) = f(x + \Delta x)$$

Fehlermaße bzgl. exakter Größe z und Störung/Approximation $\tilde{z}$

Absoluter Fehler $|z - \tilde{z}| = |\Delta z|$

Relativer Fehler $\epsilon_z := \frac{|z-\tilde{z}|}{|z|} = \frac{|\Delta z|}{|z|} \quad \Rightarrow \quad |\Delta z| = |z|\epsilon_z$

$z = \begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix}$, $|z| = \sqrt{z_1^2 + z_2^2 + z_3^2}$

Offensichtliche Frage:

Wie verhalten sich absolute/relative Fehler Δy_i, ϵ_{y_i} bzgl. der Ergebnisse y_i bei gegebenen Fehlern Δx_j, ϵ_{x_j} der Eingangsdaten?

Fehlerfortpflanzung und Kondition

Zunächst der skalare Fall ($n = m = 1$): $y = f(x)$, $\tilde{x} = x + \Delta x$, $\tilde{y} = y + \Delta y \in \mathbb{R}$

Absoluter Fehler	in x: $\lvert\Delta x\rvert = \lvert x - \tilde{x}\rvert$	in y: $\lvert\Delta y\rvert = \lvert y - \tilde{y}\rvert$
Relativer Fehler	in x: $\epsilon_x = \frac{\lvert\Delta x\rvert}{\lvert x\rvert}$	in y: $\epsilon_y = \frac{\lvert\Delta y\rvert}{\lvert y\rvert}$

Nehmen ab sofort an dass f stetig differenzierbar ist

$\rightsquigarrow$ Taylor-Approximation 1. Ordnung von $f(\tilde{x})$ an der Stelle x:

$$\tilde{y} = f(\tilde{x}) = f(x + \Delta x) \overset{\text{Taylor}}{\approx} f(x) + f'(x)(x - \tilde{x}) + \frac{f''(x)}{2}(x - \tilde{x})^2 + \dots$$

$$\Delta y = y - \tilde{y} = f(x) - f(\tilde{x}) \approx -f'(x)\Delta x$$

$$|\Delta y| \approx |f'(x)| \cdot |\Delta x|$$

$$\epsilon_y = \frac{|\Delta y|}{|y|} \approx \frac{|f'(x)| \cdot |\Delta x|}{|f(x)|} = \frac{|f'(x)| \cdot |x|}{|f(x)|} \cdot \frac{|\Delta x|}{|x|} = \underbrace{\frac{|f'(x)| \cdot |x|}{|f(x)|}}_{=:\kappa(f,x)} \cdot \epsilon_x$$

Definition: Die Größe $\kappa(f, x) = \frac{|f'(x)| \cdot |x|}{|f(x)|}$ heißt **relative Konditionszahl** von f in x. Bei $\kappa(f, x) \gg 1 \rightsquigarrow$ Verstärkung der Fehler ϵ_x der Eingangsgröße x

Fehlerfortpflanzung und Kondition

Zunächst der skalare Fall ($n = m = 1$): $y = f(x)$, $\tilde{x} = x + \Delta x$, $\tilde{y} = y + \Delta y \in \mathbb{R}$

Absoluter Fehler	in x: $\lvert\Delta x\rvert = \lvert x - \tilde{x}\rvert$	in y: $\lvert\Delta y\rvert = \lvert y - \tilde{y}\rvert$
Relativer Fehler	in x: $\epsilon_x = \frac{\lvert\Delta x\rvert}{\lvert x\rvert}$	in y: $\epsilon_y = \frac{\lvert\Delta y\rvert}{\lvert y\rvert}$

Nehmen ab sofort an dass f stetig differenzierbar ist

$\rightsquigarrow$ Taylor-Approximation 1. Ordnung von $f(\tilde{x})$ an der Stelle x:

$$\tilde{y} = f(\tilde{x}) = f(x + \Delta x) \overset{\text{Taylor}}{\approx} f(x) + f'(x)\Delta x$$

$$\Delta y = y - \tilde{y} = f(x) - f(\tilde{x}) \approx -f'(x)\Delta x$$

$$\lvert\Delta y\rvert \approx \lvert f'(x)\rvert \cdot \lvert\Delta x\rvert$$

$$\epsilon_y = \frac{\lvert\Delta y\rvert}{\lvert y\rvert} \approx \frac{\lvert f'(x)\rvert \cdot \lvert\Delta x\rvert}{\lvert f(x)\rvert} = \frac{\lvert f'(x)\rvert \cdot \lvert x\rvert}{\lvert f(x)\rvert} \cdot \frac{\lvert\Delta x\rvert}{\lvert x\rvert} = \underbrace{\frac{\lvert f'(x)\rvert \cdot \lvert x\rvert}{\lvert f(x)\rvert}}_{=:\kappa(f,x)} \cdot \epsilon_x$$

Definition: Die Größe $\kappa(f, x) = \frac{\lvert f'(x)\rvert \cdot \lvert x\rvert}{\lvert f(x)\rvert}$ heißt **relative Konditionszahl** von f in x. Bei $\kappa(f, x) \gg 1 \rightsquigarrow$ Verstärkung der Fehler ϵ_x der Eingangsgröße x

Fehlerfortpflanzung und Kondition

Allgemeiner Fall: $y = f(x)$, $\tilde{x} = x + \Delta x \in \mathbb{R}^n$, $\tilde{y} = y + \Delta y \in \mathbb{R}^m$

	in x_i:	in y_j:
Absoluter Fehler	$\lvert\Delta x_i\rvert = \lvert x_i - \tilde{x}_i\rvert$	$\lvert\Delta y_j\rvert = \lvert y_j - \tilde{y}_j\rvert$
Relativer Fehler	$\epsilon_{x_i} = \frac{\lvert\Delta x_i\rvert}{\lvert x_i\rvert}$	$\epsilon_{y_j} = \frac{\lvert\Delta y_j\rvert}{\lvert y_j\rvert}$

Lineares Fehlerfortpflanzungsgesetz

Für $j = 1 : m$:

$$\lvert\Delta y_j\rvert \approx \sum_{i=1}^{n} \left| \frac{\partial f_j(x)}{\partial x_i} \cdot \Delta x_i \right|, \qquad \epsilon_{y_j} = \left| \frac{\Delta y_j}{y_j} \right| \approx \sum_{i=1}^{n} \underbrace{\left| \frac{\partial f_j(x)}{\partial x_i} \cdot \frac{x_i}{f_j(x)} \right|}_{=\kappa_{ij}} \cdot \epsilon_{x_i}$$

Definition: Die Größen κ_{ij} heißen **relative Konditionszahlen** von f_j. Sie zeigen an wie sensitiv $y_j = f_j(x_1, \ldots, x_n)$ ist bzgl. Änderungen in x_i:

$\kappa_{ij} \gg 1 \quad \Rightarrow$ große relative Fehler ϵ_{y_j} bei kleinen Fehlern ϵ_{x_i}

Fehlerfortpflanzung und Kondition

Kondition

Die Kondition ist eine **dem mathematischen Problem innewohnende Eigenschaft**.

Hat ein Problem sehr große Konditionszahl(en), $\kappa \gg 1$, nennt man es „schlecht konditioniert“: Kleine relative Fehler in Eingangsdaten führen zu großen relativen Fehlern im Resultat.

Mit $\kappa \approx 1$ liegt ein „gut konditioniertes“ Problem vor: Kleine Fehler in Eingangsdaten führen zu ~~großen~~ kleinen relativen Fehlern im Resultat, $\epsilon_y \approx \epsilon_x$

Die Kondition misst den unvermeidbaren Fehler, der durch das Problem selbst gegeben ist und hängt stark von den Daten ab.

$Ax = b$

Beispiele:

- Subtraktion $a - b$ ist schlecht konditionierte Operation für $a \approx b$
- Multiplikation $a \cdot b$ ist gut konditioniert

Fehlerfortpflanzung und Kondition

Verbindung zu Fehlerrechnung (vgl. Mathe II):

Berechnung physikalischer/technischer Größe $y = f(x) = f(x_1, \ldots, x_n)$ aus Messdaten: $x_i = \overline{x}_i \pm \Delta x_i$, $\overline{x}_i \hat{=}$Mittelwert, $\Delta x_i \hat{=}$Messfehler

- Auswertung an Mittelwerten $\overline{y} = f(\overline{x}_1, \ldots, \overline{x}_n)$
- Schätzung der absoluten Abweichung gemäß linearen Fehlerfortpflanzungsgesetz:
  $|\Delta y| \approx \sum_{i=1}^{n} \left| \frac{\partial f(\overline{x})}{\partial x_i} \cdot \Delta x_i \right|$
  $\Rightarrow$ gesuchte Größe im Bereich $y \in [\overline{y} - \Delta y, \overline{y} + \Delta y]$
- Schätzung der relativen Abweichung $\epsilon_y = \frac{|\Delta y|}{|\overline{y}|}$ analog.
- Erlaubt Berechnung der relativen Konditionszahlen $\left| \frac{\partial f(\overline{x})}{\partial x_i} \cdot \frac{\overline{x}_i}{f(\overline{x})} \right|$

→ Bsp im Seminar

Fehlerfortpflanzung und Kondition

Numerische Stabilität von Algorithmen

Ist ein **Problem** $y = f(x)$ **gut konditioniert** und

existiert zusätzlich auch ein numerisches Berechnungsverfahren, bei dem die **relativen Fehler nicht zusätzlich stark vergrößert werden**, so spricht man von einem **numerisch stabilen** Algorithmus.

Ein Algorithmus, der trotz kleiner Konditionszahl zu vergrößerten relativen Fehlern ϵ_y im Resultat führen kann, heißt **numerisch instabil**.

- Gut konditioniertes Problem ($\kappa \approx 1$) und stabiler Algorithmus $\rightsquigarrow$ zuverlässige Resultate
- Schlecht konditioniertes Problem (κ groß) oder instabiler Algorithmus $\rightsquigarrow$ keine sicheren Ergebnisse erwartbar
- Design und Entwicklung stabiler Algorithmen:
  - Betrachte Algorithmus als Folge vieler zur berechnender Teilprobleme/Operationen
  - Stabiler Algorithmus: wähle gut konditionierte Teilprobleme/Operationen

Ende Vorlesung 1

Übung zur Vorbereitung für nächste Vorlesung:

- Wiederholung lineare Algebra: lineare (Un)abhängigkeit von Vektoren, Determinante einer Matrix, Lösbarkeit linearer Gleichungssysteme, Gauß-Algorithmus (!), ...