Numerical Methods for Computer Science

Robin Bacher, Janis Hutz
https://github.com/janishutz/eth-summaries

26. September 2025

TITLE PAGE COMING SOON

"*Denken vor Rechnen*"
- Vasile Grudinaru, 2025

HS2025, ETHZ
Summary of the Script and Lectures

Inhaltsverzeichnis

1 Einführung

1.1 Rundungsfehler

Absoluter & Relativer Fehler — **Definition 1.1**

- ***Absoluter Fehler***: $||\widetilde{x} - x||$
- ***Relativer Fehler***: $\frac{||\widetilde{x} - x||}{||x||}$ für $||x|| \neq 0$

wobei $\widetilde{x}$ eine Approximation an $x \in \mathbb{R}$ ist

Rundungsfehler entstehen durch die (verhältnismässig) geringe Präzision die man mit der Darstellung von Zahlen auf Computern erreichen kann. Zusätzlich kommt hinzu, dass durch Unterläufe (in diesem Kurs ist dies eine Zahl die zwischen 0 und der kleinsten darstellbaren, positiven Zahl liegt) Präzision verloren gehen kann.

Überläufe hingegen sind konventionell definiert, also eine Zahl, die zu gross ist und nicht mehr dargestellt werden kann.

Auslöschung — **Bemerkung 1.9**

Bei der Subtraktion von zwei ähnlich grossen Zahlen kann es zu einer Addition der Fehler der beiden Zahlen kommen, was dann den relativen Fehler um einen sehr grossen Faktor vergrössert. Die Subtraktion selbst hat einen vernachlässigbaren Fehler

Beispiel 1.18: *(Ableitung mit imaginärem Schritt)* Als Referenz in Graphen wird hier oftmals die Implementation des Differenzialquotienten verwendet.

Der Trick hier ist, dass wir mit Komplexen Zahlen in der Taylor-Approximation einer glatten Funktion in x_0 einen rein imaginären Schritt durchführen können:

$$f(x_0 + ih) = f(x_0) + f'(x_0)ih - \frac{1}{2}f''(x_0)h^2 - iC \cdot h^3 \text{ für } h \in \mathbb{R} \text{ und } h \to 0$$

Da $f(x_0)$ und $f''(x_0)h^2$ reell sind, verschwinden die Terme, wenn wir nur den Imaginärteil des Ausdruckes weiterverwenden. Nach weiteren Vereinfachungen und Umwandlungen erhalten wir

$$f'(x_0) \approx \frac{\text{Im}(f(x_0 + ih))}{h}$$

Falls jedoch hier die Auswertung von $\text{Im}(f(x_0 + ih))$ nicht exakt ist, so kann der Fehler beträchtlich sein.

Beispiel 1.20: *(Konvergenzbeschleunigung nach Richardson)*

$$\begin{aligned} yf'(x) &= yd\left(\frac{h}{2}\right) + \frac{1}{6}f'''(x)h^2 + \frac{1}{480}f^{(s)}h^4 + \ldots - f'(x) \\ &= -d(h) - \frac{1}{6}f'''(x)h^2 + \frac{1}{120}f^{(s)}(x)h^n \Leftrightarrow 3f'(x) \\ &= 4d\left(\frac{h}{2}\right)d(h) + \mathcal{O}\left(h^4\right) \Leftrightarrow \end{aligned}$$

Schema

$$d(h) = \frac{f(x+h) - f(x-h)}{2h}$$

wobei im Schema dann

$$R_{l,0} = d\left(\frac{h}{2^l}\right)$$

und

$$R_{l,k} = \frac{4^k \cdot R_{l,k-1} - R_{l-1,k-1}}{4^k - 1}$$

und $f'(x) = R_{l,k} + C \cdot \left(\frac{h}{2^l}\right)^{2k+2}$

1.2 Rechenaufwand

In NumCS wird die Anzahl elementarer Operationen wie Addition, Multiplikation, etc benutzt, um den Rechenaufwand zu beschreiben. Wie in Algorithmen und * ist auch hier wieder $\mathcal{O}(\dots)$ der Worst Case. Teilweise werden auch andere Funktionen wie $\sin, \cos, \sqrt{\dots}, \dots$ dazu gezählt.

Die Basic Linear Algebra Subprograms (= BLAS), also grundlegende Operationen der Linearen Algebra, wurden bereits stark optimiert und sollten wann immer möglich verwendet werden und man sollte auf keinen Fall diese selbst implementieren.

Dieser Kurs verwendet `numpy`, `scipy`, `sympy` (collection of implementations for symbolic computations) und `matplotlib`. Dieses Ecosystem ist eine der Stärken von Python und ist interessanterweise zu einem Grossteil nicht in Python geschrieben, da dies sehr langsam wäre.

1.3 Rechnen mit Matrizen

Name	Operation	Mult	Add	Komplexität
Skalarprodukt	$x^H y$	n	$n-1$	$\mathcal{O}(n)$
Tensorprodukt	xy^H	nm	0	$\mathcal{O}(mn)$
Matrix $\times$ Vektor	Ax	mn	$(n-1)m$	$\mathcal{O}(mn)$
Matrixprodukt	AB	mnp	$(n-1)mp$	$\mathcal{O}(mnp)$

Das Matrixprodukt kann mit dem Strassen Algorithmus mithilfe der Block-Partitionierung in $\mathcal{O}\left(n^{\log_2(7)}\right)$

2 Interpolation