HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Mathematisches Institut
Lehrstuhl für Angewandte Mathematik
Prof. Dr. Christiane Helzel

Skript zur Vorlesung

# Numerik 1

# Vorwort

Die Vorlesung zur Numerik 1 wurde im Sommersemester 2014 von Frau Prof. Dr. Christiane Helzel am Mathematischen Institut der Heinrich-Heine-Universität Düsseldorf gehalten. Dieses Vorlesungsskript entstand aus einer Mitschrift der Studentin Carolin Albrecht. Eine Überarbeitung erfolgte im Sommersemester 2016 von Maximilian Schneiders. Es enthält außerdem einige Ergänzungen zur Vorlesung.

Dieses Skript soll stetig erweitert und verbessert werden. Da Fehler, sowohl inhaltlich als auch sprachlich, nicht ausgeschlossen sind, sind Hinweise auf diese und andere Verbesserungsvorschläge, um die Qualität des Skripts zu verbessern, ausdrücklich erwünscht und können an maximilian.schneiders@uni-duesseldorf.de gesendet werden.

Als weitere Literatur zu dieser Vorlesung dienen die Lehrbücher Stoer/Bulirsch, Numerische Mathematik 1, Springer und P. Deuflhard, A. Hohmann, Numerische Mathematik 1, de Gruyter. Für Beispiele in MATLAB kann das Buch Cleve B. Moler, Numerical Computing with MATLAB, SIAM hinzugezogen werden.

Zuletzt bearbeitet: 25. Mai 2016

# Inhaltsverzeichnis

# 1 Fehleranalyse

Ein Computer kann Zahlen nur mit endlich vielen Ziffern darstellen. Dies führt zwangsläufig zu Fehlern in numerischen Algorithmen. In diesem Kapitel beschäftigen wir uns mit der Darstellung von Zahlen in einem Computer sowie mit den dadurch verbundenen Problemen von numerischen Algorithmen.

## 1.1 Gleitkommadarstellung

Zahlen werden in einem Computer mit Hilfe von normalisierten Gleitkommazahlen realisiert. Es sei $x \in \mathbb{R}$ eine reelle Zahl. Dann lässt sich diese Zahl in der Form

$$x = \pm m \cdot b^{\pm e}$$

schreiben. Dabei bezeichen wir mit $m$ die *Mantisse*, mit $b$ die *Basis* (im Binärsystem ist $b = 2$ und im Dezimalsystem ist $b = 10$) und mit $e$ den *Exponenten*. Durch den ANSI/IEEE Standard (American National Standard Institute, Institute of Electrical and Electronics Engineers, seit 1985) werden im Computer Darstellungen für binäre Gleitkommazahlen definiert, das heißt wir betrachten Zahlen der Form

$$x = \pm(1 + f)2^{c-1023},$$

wobei die Binärdarstellung von $f$ maximal 52 Bits benutzt, das heißt es ist

$$0 \leq 2^{52} \cdot f < 2^{52} \quad \text{mit} \quad 0 \leq f < 1$$

und

$$f = f_1 \cdot 2^{-1} + f_2 \cdot 2^{-2} + \cdots + f_{52} \cdot 2^{-52}$$

mit $f_i \in \{0, 1\}$. Die Zahl $c$ wird wie folgt charakterisiert: Es werden 11 Bits für den Exponenten vergeben, das heißt wir erhalten

$$c = c_0 \cdot 2^0 + c_1 \cdot 2^1 + \cdots + c_{10} \cdot 2^{10} \quad \text{mit} \quad c_i \in \{0, 1\},$$

also $0 \leq c \leq 2^0 + 2^1 + \cdots + 2^{10} = 2047$ und damit

$$-1023 \leq e \leq 1024.$$

Die Werte $e = -1023$ und $e = 1024$ werden zur Speicherung besonderer „Zahlen“ verwendet:

- $e = 1024$ und $f = 0$: Inf (infinity).
- $e = 1024$ und $f \neq 0$: NaN (not a number).
- $e = -1023$ und $f = 0, c = 0$: Darstellung der Null.

```
>> 1/Inf
ans =
     0

>> Inf+Inf
ans =
   Inf

>> 0/0
ans =
   NaN

>> Inf-Inf
ans =
   NaN
```

Beispiele in MATLAB

Für alle anderen Zahlen gilt $-1022 \leq e \leq 1023$. Gleitkommazahlen (mit doppelter Genauigkeit) werden mit insgesamt 64 Bits gespeichert:

- 52 Bits für die Mantisse,
- 11 Bits für den Exponenten,
- 1 Bit für das Vorzeichen.

*Bemerkung*. Die Verwendung der Gleitkommadarstellung im numerischen Rechnen ist wesentlich, um Zahlen sehr unterschiedlicher Größe bearbeiten zu können. Des Weiteren liefert die Darstellung von $m$ eine Beschränkung an die Genauigkeit und die Darstellung von $e$ eine Beschränkung der Größenordnung.

Es gelte nun $-1022 \leq e \leq 1023$. Dann gilt für die größte/kleinste Gleitkommazahl

$$x_{\max,\min} = \pm[1 + 2^{-1} + 2^{-2} + \cdots + 2^{-52}] \cdot 2^{1023} \approx \pm 1{,}8 \cdot 10^{308}.$$

Für die kleinste positive/größte negative Gleitkommazahl erhalten wir

$$x_{\text{posmin}} = [1 + 0 \cdot 2^{-1} + 0 \cdot 2^{-2} + \cdots + 0 \cdot 2^{-52}] \cdot 2^{-1022} \approx 2{,}2 \cdot 10^{-308},$$
$$x_{\text{negmax}} = -2^{-1022} \approx -2{,}2 \cdot 10^{-308}.$$

Mit $A = A(b, f, c)$ bezeichnen wir das *numerische Gleitkommagitter*. Die Menge $A$ ist die Menge der in der Form $x = \pm(1 + f) \cdot b^{c-1023}$ darstellbaren Maschinenzahlen. Dabei beschreibt $f$ die Darstellung der Mantisse und $c$ die Darstellung des Exponenten. Der zulässige Bereich wird definiert als

$$D := [x_{\min}, x_{\text{negmax}}] \cup \{0\} \cup [x_{\text{posmin}}, x_{\max}].$$

```
>> realmin % kleinste positive Zahl
ans =
  2.2251e-308

>> realmax % größte positive Zahl
ans =
  1.7977e+308

>> 2*2^1023 % Overflow
ans =
   Inf

>> 2^-1022
ans =
  2.2251e-308

>> eps*2^-1022
ans =
  4.9407e-324

>> eps*2^-1023
ans =
     0
```

Beispiele in MATLAB

Wir definieren auf $D$ die Rundungsoperation $\mathrm{rd} : D \to A$ mit

$$|x - \mathrm{rd}(x)| = \min_{y \in A}(|x - y|) \quad \text{für alle} \quad x \in D.$$

Beispiel:

$$\mathrm{rd}(x) = \mathrm{sign}(x) \cdot \begin{cases} (1 + f_1 \cdot 2^{-1} + \cdots + f_{52} \cdot 2^{-52}) \cdot 2^{c-1023}, & f_{53} = 0, \\ (1 + f_1 \cdot 2^{-1} + \cdots + f_{52} \cdot 2^{-52} + 2^{-52}) \cdot 2^{c-1023}, & f_{53} = 1. \end{cases}$$

(Man nimmt dabei an, dass der Rechner intern mit mehr Stellen arbeitet.) Dann gilt:

- absoluter Rundungsfehler: $|x - \mathrm{rd}(x)| \leq \frac{1}{2} \cdot 2^{-52} \cdot 2^e$,
- relativer Rundungsfehler: $\left|\frac{x-\mathrm{rd}(x)}{x}\right| \leq \frac{1}{2} \cdot \frac{2^{-52}}{(1+f)\cdot 2^e} < 2^{-53}$,
- maximaler Rundungsfehler $\frac{\mathrm{eps}}{2} = \max\limits_{x\in D, x\neq 0} \left|\frac{\mathrm{rd}(x)-x}{x}\right|$.

Dabei bezeichnen wir mit eps die *Maschinengenauigkeit*. Im IEEE-Format gilt

$$\frac{\mathrm{eps}}{2} = \frac{1}{2} \cdot 2^{-52} = 2^{-53} \approx 1{,}11 \cdot 10^{-16} \quad \text{bzw.} \quad \mathrm{eps} \approx 2{,}2204 \cdot 10^{-16}.$$

Die Maschinengenauigkeit gibt den Abstand von 1 zur nächstgelegenen Gleitkommazahl an bzw. den relativen Abstand zwischen den Gleitkommazahlen.

```
format long;
myeps = 1.d0;

while (1.d0+myeps)>1.d0
    myeps = myeps/2.d0;
end
2*myeps
```

**MATLAB-Code 1:** Berechnung der Maschinengenauigkeit in MATLAB

Für $x \in D$ gilt $\mathrm{rd}(x) = x(1+\varepsilon)$ mit $|\varepsilon| \leq \mathrm{eps}$. Die Grundoperationen $* \in \{+, -, \cdot, /\}$ werden im Computer durch Maschinenoperationen $\circledast \in \{\oplus, \ominus, \odot, \oslash\}$ ersetzt. Diese sind so definiert: Für $x, y \in A$ und $x * y \in D$ gilt $x \circledast y = (x * y)(1+\varepsilon)$ mit $|\varepsilon| \leq \mathrm{eps}$.

*Bemerkung.* Für $x, y \in A$ gilt:

(i) $(x \oplus y) \oplus z \neq x \circledast (y \oplus z)$ (kein Assoziativgesetz),

(ii) $(x \oplus y) \odot z \neq (x \odot z) \oplus (y \odot z)$ (kein Distributivgesetz),

(iii) $x \oplus y = x$ für $|y| \leq \frac{|x|}{2}\mathrm{eps}$.

Aus (iii) lässt sich die Größe der Maschinengenauigkeit eps experimentell bestimmen (vergleiche MATLAB-Code 1).

## 1.2 Konditionierung numerischer Aufgaben

In diesem Abschnitt beschäftigen wir uns mit der Frage, wie sich Störungen in der Eingabegröße auf das Resultat auswirken, unabhängig vom gewählten Algorithmus. Eine numerische Aufgabe wird als *gut konditioniert* bezeichnet, wenn eine kleine Störung der

Eingabedaten auch nur eine kleine Änderung im Ergebnis zur Folge hat. Ansonsten wird die numerische Aufgabe als *schlecht konditioniert* bezeichnet. Unter einer numerischen Aufgabe versteht man die Berechnung endlich vieler Größen $y_i$ $(i = 1, \ldots, n)$ aus Größen $x_j$ $(j = 1, \ldots, m)$ mittels einer funktionalen Vorschrift $y_i = f_i(x_1, \ldots, x_m)$. Schreibweise: $y = f(x)$, $x = (x_1, \ldots, x_m)^T$, $y = (y_1, \ldots, y_m)^T$, $f = (f_1, \ldots, f_m)^T$.

```
clear all;

A = [1.2969 0.8648; 0.2161 0.1441];
b1 = [0.8642; 0.1440];

A\b1
%%
b2 = [0.86419999; 0.14400001];
A\b2
```

**MATLAB-Code 2:** Beispiel für eine schlecht konditionierte Aufgabe in MATLAB

Der obige MATLAB-Code berechnet die Lösung eines linearen Gleichungssystems. Im ersten Fall ist die Lösung gegeben durch $(x, y) = (2, -2)^T$. Durch geringe Störungen der rechten Seite ergibt sich die Lösung $(x, y) = (0{,}9911, -0{,}4870)^T$, also ein völlig anderes Ergebnis. Diese Aufgabe ist somit schlecht konditioniert.

**Definition 1.1.** Bei Verwendung fehlerhafter Eingabedaten $x_j + \Delta x_j$ (z.B. aufgrund des Rundungsfehlers) ergeben sich fehlerhafte Resultate $y_i + \Delta y_i$. Wir bezeichnen $|\Delta y_i|$ als den *absoluten Fehler* und $\left|\frac{\Delta y_i}{y_i}\right|$ für $y_i \neq 0$ als den *relativen Fehler*.

**Definition 1.2** (Landau-Symbole)**.** Für Funktionen $g(t)$ und $h(t)$ der Variablen $t \in \mathbb{R}_+$ bedeutet die Schreibweise

$$g(t) = \mathcal{O}(h(t)) \quad \text{für} \quad t \to 0,$$

dass für kleine $t \in ]0, t_0]$ mit einer Konstanten $c \geq 0$ gilt:

$$|g(t)| \leq c \cdot |h(t)|.$$

Entsprechend bedeutet

$$g(t) = o(h(t)) \quad \text{für} \quad t \to 0,$$

dass für kleine $t \in ]0, t_0]$ mit einer Funktion $c(t) \to 0$ für $t \to 0$ gilt:

$$|g(t)| \leq c(t) \cdot |h(t)|.$$

## Differentielle Fehleranalyse

Es sei $|\Delta x_j| \ll |x_j|$ (relativ kleiner Datenfehler) und $f_i = f_i(x_1, \ldots, x_m)$ zweimal stetig partiell differentierbar nach den Argumenten $x_j$. Dann gilt mit Taylorentwicklung:

$$\Delta y_i = f_i(x + \Delta x) - f_i(x) = \sum_{j=1}^{m} \frac{\partial f_i}{\partial x_j}(x)\Delta x_j + R_i^f(x, \Delta x), \quad i = 1, \ldots, m, \qquad (*)$$

wobei $R_i^f(x, \Delta x) = \mathcal{O}(|\Delta x|^2)$ das Restglied ist. Aus $(*)$ folgt in erster Näherung

$$\Delta y_i \approx \sum_{j=1}^{m} \frac{\partial f_i}{\partial x_j}(x)\Delta x_j,$$

das heißt Gleichheit gilt in erster Näherung bis auf einen Term der Ordnung $\mathcal{O}\left(|\Delta x|^2\right)$. Für den relativen Fehler (komponentenweise) folgt mit $y_i = f_i(x)$ also

$$\frac{\Delta y_i}{y_i} \approx \sum_{j=1}^{m} \frac{\partial f_i}{\partial x_j}(x)\frac{\Delta x_j}{y_i} = \sum_{j=1}^{m} \frac{\partial f_i}{\partial x_j}(x)\frac{x_j}{f_i(x)}\frac{\Delta x_j}{x_j}.$$

**Definition 1.3.** Die Größen

$$K_{i,j}(x) := \frac{\partial f_i}{\partial x_j}(x)\frac{x_j}{f_i(x)}$$

heißen *relative Konditionszahlen* der Funktion $f$ im Punkt $x$. Sie sind ein Maß dafür, wie sich kleine relative Fehler in den Ausgangsdaten im Ergebnis auswirken. Man nennt die Aufgabe, $y = f(x)$ aus $x$ zu berechnen, s*chlecht konditioniert*, wenn $|K_{i,j}(x)| \gg 1$ ist, andernfalls *gut konditioniert*.

## Grundoperationen

1) Wir betrachten für $x_1, x_2 \in \mathbb{R}$, $x_1 \cdot x_2 \neq 0$ die Addition $y = f(x_1, x_2) = x_1 + x_2$. Dann gilt

$$K_1 = \frac{\partial f}{\partial x_1} \cdot \frac{x_1}{f} = 1 \cdot \frac{x_1}{x_1 + x_2} = \frac{1}{1 + \frac{x_2}{x_1}},$$
$$K_2 = \frac{\partial f}{\partial x_2} \cdot \frac{x_2}{f} = 1 \cdot \frac{x_2}{x_1 + x_2} = \frac{1}{1 + \frac{x_1}{x_2}}.$$

Für $\frac{x_1}{x_2} \approx -1$ (also für die Subtraktion zweier fast gleich großer Zahlen) ist die Addition schlecht konditioniert.

**Definition 1.4** (Auslöschung)**.** Unter *Auslöschung* versteht man den Verlust an wesentlichen Dezimalstellen bei der Subtraktion von Zahlen gleichen Vorzeichens. Dies ist gefährlich im Fall, dass eine oder beide Zahlen keine Maschinenzahlen sind und diese vor Ausführung der Operation gerundet werden. Bei der Subtraktion von Maschinenzahlen ist Auslöschung natürlich unschädlich.

*Beispiel* (Dezimale Gleitpunktrechnung mit vier signifikanten Ziffern). Gegeben seien

$$x_1 = 0{,}11258762 \cdot 10^2, \quad x_2 = 0{,}11244891 \cdot 10^2.$$

Da wir nur vier signifikante Ziffern betrachten, erhalten wir

$$\mathrm{rd}(x_1) = 0{,}1125 \cdot 10^2, \quad \mathrm{rd}(x_2) = 0{,}1124 \cdot 10^2$$

sowie jeweils einen absoluten Fehler von

$$|\Delta x_1| = 0{,}001238, \quad |\Delta x_2| = 0{,}004891.$$

Der relative Fehler in der Eingabe beträgt somit

$$\left|\frac{\Delta x_1}{x_1}\right| = 1{,}099 \cdot 10^{-4}, \quad \left|\frac{\Delta x_2}{x_2}\right| = 4{,}3495 \cdot 10^{-4}.$$

Nun gilt

$$x_1 + x_2 = 0{,}22503653 \cdot 10^2, \quad \mathrm{rd}(x_1) + \mathrm{rd}(x_2) = 0{,}2250 \cdot 10^2$$

und

$$x_1 - x_2 = 0{,}13871 \cdot 10^{-1}, \quad \mathrm{rd}(x_1) - \mathrm{rd}(x_2) = 0{,}2000 \cdot 10^{-1}.$$

Wir berechnen für den relativen Fehler der Subtraktion

$$\left|\frac{(x_1 - x_2) - (\mathrm{rd}(x_1) - \mathrm{rd}(x_2))}{x_1 - x_2}\right| = 0{,}44185.$$

Mit $f(x_1, x_2) = x_1 - x_2$ erhalten wir für die relativen Konditionszahlen die Werte

$$K_1 = \left|\frac{\partial f}{x_1} \cdot \frac{x_1}{f}\right| = \left|\frac{x_1}{x_1 - x_2}\right| = 811{,}67, \quad K_2 = \left|\frac{\partial f}{x_2} \cdot \frac{x_2}{f}\right| = \left|-\frac{x_2}{x_1 - x_2}\right| = 810{,}67,$$

also

$$K_1 \left|\frac{\Delta x_1}{x_1}\right| + K_2 \left|\frac{\Delta x_2}{x_2}\right| = 0{,}44139.$$

Die Subtraktion fast gleich großer Zahlen ist also sehr schlecht konditioniert. Im Gegensatz dazu gilt für die Addition

$$K_1 = 0{,}5003, \quad K_2 = 0{,}4997.$$

2) Betrachten wir nun die Multiplikation $y = f(x_1, x_2) = x_1 \cdot x_2$. Dann gilt

$$K_1 = \frac{\partial f}{\partial x_1} \cdot \frac{x_1}{f} = \frac{x_1 x_2}{x_1 x_2} = 1,$$
$$K_2 = \frac{\partial f}{\partial x_2} \cdot \frac{x_2}{f} = \frac{x_2 x_1}{x_2 x_1} = 1.$$

Das heißt die Multiplikation ist gut konditioniert. Damit ist auch die Division gut konditioniert.

### Lösung quadratischer Aufgaben

Betrachte für $p, q \in \mathbb{R}$, $q \neq 0$ und $\frac{p^2}{4} - q > 0$ die quadratische Gleichung

$$y^2 + py + q = 0.$$

Die Lösungsformel für diese Gleichung liefert die Nullstellen

$$y_{1,2} = -\frac{p}{2} \pm \sqrt{\left(\frac{p}{2}\right)^2 - q}.$$

Außerdem gilt $y_1 + y_2 = -p$ und $y_1 y_2 = q$ (Satz von Vieta). Man kann nun zeigen (Übung), dass die Berechnung von $y_1$ und $y_2$ für $y_1 \approx y_2$ schlecht konditioniert ist.

## 1.3 Stabilität numerischer Algorithmen

Betrachte eine numerische Aufgabe $y = f(x)$ mit $f : \mathbb{R}^m \to \mathbb{R}^n$. Ein *Verfahren* oder *Algorithmus* zur Berechnung von $y$ aus $x$ ist eine endliche (oder abzählbar unendliche) Folge von „elementaren" Abbildungen $\varphi^{(k)}$ (z. B. arithmetische Grundoperationen), die durch sukzessive Anwendung einen Näherungswert $\tilde{y}$ zu $y$ liefert.

**Definition 1.5.** Bei der Durchführung des Algorithmus (auf einer Rechenanlage) treten in jedem Schritt Fehler auf (z. B. Rundungsfehler), die sich bis zum Ende der Rechnung akkumulieren können. Der Algorithmus wird *stabil* (oder gutartig) genannt, wenn die im Verlauf der Ausführung akkumulierten Fehler den durch die Konditionierung der Aufgabe bedingten unvermeidbaren Problemfehler nicht übersteigen.

Beispielsweise sei das Polynom $p(x) = (x-1)^7$ gegeben. Abbildung 1.1 zeigt den Plot von $p$ in einer kleinen Umgebung der 7-fachen Nullstelle $x = 1$. Während die Auswertung der Faktorisierung $p(x) = (x-1)^7$ einen stabilen Algorithmus liefert, ist der

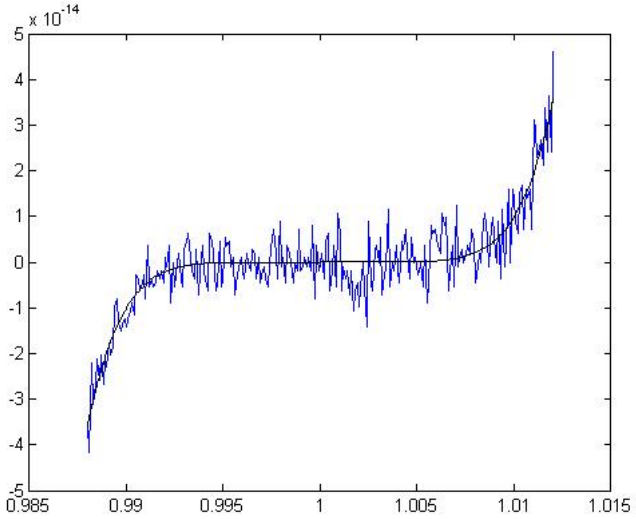


**Abbildung 1.1:** Plot eines Polynoms siebten Grades, das auf zwei verschiedene Weisen berechnet wurde.

Algorithmus, der das Polynom in der Form

$$p(x) = \sum_{k=0}^{7} a_k x^k = x^7 - 7x^6 + 21x^5 - 35x^4 + 35x^3 - 21x^2 + 7x - 1$$

auswertet, sehr instabil.

```
x = 0.988:0.0001:1.012;
y1 = x.^7 - 7*x.^6 + 21*x.^5 -35*x.^4 + 35*x.^3 -21*x.^2 + 7*x -1;
plot(x,y1)
%%

y2 = (x-1).^7;
hold on;
plot(x,y2,'k')
```

**MATLAB-Code 3:** Berechnung eines Polynoms siebten Grades auf zwei verschiedene Weisen.

Eine der Hauptaufgaben der numerischen Mathematik ist es, für die in den Anwendungen auftretenden Aufgaben, stabile Lösungsalgorithmen zu finden. Kommen wir nun nochmal zurück zur Lösung quadratischer Gleichungen.

Für $q \neq 0$, $p > 0$ und $q < \frac{p^2}{4}$ sind die Lösungen von $y^2 + py + q = 0$ gegeben durch

$$y_{1,2} = -\frac{p}{2} \pm \sqrt{\left(\frac{p}{2}\right)^2 - q}.$$

Betrachte nun den Fall $|\frac{y_2}{y_1}| \gg 1$, das heißt $q \ll \frac{p^2}{4}$. Damit ist die Berechnung von $y_2$ gut konditioniert. Ein Algorithmus zur Berechnung von $y_1$ und $y_2$ ist dann wie folgt:

1. $u = \left(\frac{p}{2}\right)^2$,
2. $v = u - q$,
3. $w = \sqrt{v} \quad (\geq 0)$.

Da $p > 0$ ist, wird zur Vermeidung von Auslöschung zunächst

$$y_2 = -\frac{p}{2} - w = f(p, w)$$

berechnet.

### Fehlerfortpflanzung

Betrachten wir als erstes den relativen Fehler von $y_2$. Dann gilt mit

$$\frac{\partial f}{\partial p} \cdot \frac{p}{f} = -\frac{1}{2} \cdot \frac{p}{-\frac{p}{2} - w} = \frac{1}{1 + \frac{2w}{p}} \quad \text{und} \quad \frac{\partial f}{\partial w} \cdot \frac{w}{f} = -1 \cdot \frac{w}{-\frac{p}{2} - w} = \frac{1}{\frac{p}{2w} + 1}$$

die Abschätzung

$$\begin{aligned} \left|\frac{\Delta y_2}{y_2}\right| &\leq \left|\frac{\partial f}{\partial p} \cdot \frac{p}{f}\right| \left|\frac{\Delta p}{p}\right| + \left|\frac{\partial f}{\partial w} \cdot \frac{w}{f}\right| \left|\frac{\Delta w}{w}\right| \\ &= \left|\frac{1}{1 + \frac{2w}{p}}\right| \left|\frac{\Delta p}{p}\right| + \left|\frac{1}{1 + \frac{p}{2w}}\right| \left|\frac{\Delta w}{w}\right| \\ &< 1 \cdot \left|\frac{\Delta p}{p}\right| + 1 \cdot \left|\frac{\Delta w}{w}\right|. \end{aligned}$$

Das heißt die Fehlerforpflanzung in diesem Schritt ist akzeptabel.

Für die Berechnung von $y_1$ betrachten wir jetzt zwei Varianten:

- Variante A: Berechne $y_1 = -\frac{p}{2} + w$ nach der Lösungsformel,
- Variante B: Berechne $y_1 = \frac{q}{y_2}$ mit Hilfe des Satzes von Vieta.

Für die erste Variante gilt $y_1 = f(p, w) = -\frac{p}{2} + w$. Mit

$$\frac{\partial f}{\partial p} \cdot \frac{p}{f} = -\frac{1}{2} \cdot \frac{p}{-\frac{p}{2} + w} = \frac{1}{1 - \frac{2w}{p}} \quad \text{und} \quad \frac{\partial f}{\partial w} \cdot \frac{w}{f} = 1 \cdot \frac{w}{-\frac{p}{2} + w} = \frac{1}{1 - \frac{p}{2w}}$$

ergibt sich für den relativen Fehler von $y_1$ dann

$$\left|\frac{\Delta y_1}{y_1}\right| \leq \left|\frac{1}{1-\frac{2w}{p}}\right| \left|\frac{\Delta p}{p}\right| + \left|\frac{1}{1-\frac{p}{2w}}\right| \left|\frac{\Delta w}{w}\right|.$$

Da $\frac{1}{1-\frac{p}{2w}} \gg 1$ und $\frac{1}{1-\frac{2w}{p}} \gg 1$, ist dieser Algorithmus für $q \ll \frac{p^2}{4}$ sehr instabil.

Für die zweite Variante hingegen setzen wir $y_1 = f(q, y_2) = \frac{q}{y_2}$. Diesmal berechnen wir

$$\frac{\partial f}{\partial q} \cdot \frac{q}{f} = \frac{1}{y_2} \cdot \frac{q}{\frac{q}{y_2}} = 1 \quad \text{und} \quad \frac{\partial f}{\partial y_2} \cdot \frac{y_2}{f} = \frac{q}{y_2^2} \cdot \frac{y_2}{\frac{q}{y_2}} = 1.$$

Dann lässt sich der relative Fehler von $y_1$ abschätzen durch

$$\left|\frac{\Delta y_1}{y_1}\right| \leq \left|\frac{\Delta q}{q}\right| + \left|\frac{\Delta y_2}{y_2}\right|.$$

Da der relative Fehler von $y_2$ klein ist, ist auch der relative Fehler von $y_1$ klein. Dieser Algorithmus ist also stabil.

*Bemerkung.* Bei der Lösung quadratischer Gleichungen sollten nicht beide Wurzeln aus der Lösungsformel berechnet werden.

### Integral-Rekursion

Aufgabe: Berechne

$$I_n = \frac{1}{e} \int_0^1 x^n e^x \, \mathrm{d}x \quad \text{für} \quad n = 0, 1, \ldots, 30.$$

Partielle Integration liefert uns

$$\begin{aligned} I_n &= \frac{1}{e} \int_0^1 x^n e^x \, \mathrm{d}x \\ &= \frac{1}{e} \left[x^n e^x\right]_0^1 - \int_0^1 n x^{n-1} e^x \, \mathrm{d}x \\ &= \frac{1}{e} \left( e - n \int_0^1 x^{n-1} e^x \, \mathrm{d}x \right) \\ &= 1 - n I_{n-1}. \end{aligned}$$

Damit erhalten wir die Zwei-Term-Rekursion

$$I_n = 1 - nI_{n-1}.$$

Als Startwert berechnen wir

$$I_0 = \frac{1}{e} \int_0^1 e^x \, \mathrm{d}x = \frac{1}{e}(e-1) \approx 0{,}63212\ldots$$

Ausgehend von $I_0$ lassen sich aus dieser Formel beliebige Integrale $I_n$ für $n > 0$ berechnen.

```
I(0)  =  6.3212e-001
I(5)  =  1.4553e-001
I(10) =  8.3877e-002
I(15) =  5.9034e-002
I(16) =  5.5459e-002
I(17) =  5.7192e-002
I(18) = -2.9454e-002
I(19) =  1.5596e+000
I(20) = -3.0192e+001
I(21) =  6.3504e+002
I(22) = -1.3970e+004
```

Einige Integrale der Integral-Rekursion. Ab $I_{18}$ bekommt man völlig falsche Werte.

Wir erhalten

$$I_n = \frac{1}{e}\int_0^1 x^n e^x \,\mathrm{d}x \geq 0, \quad \text{da} \quad x^n e^x > 0 \quad \text{für alle} \quad x \in (0,1).$$

Wegen $e^x < e$ für alle $x \in (0,1)$ können wir weiterhin

$$I_n = \frac{1}{e}\int_0^1 x^n e^x \,\mathrm{d}x \leq \frac{1}{e} e \int_0^1 x^n \,\mathrm{d}x = \left[\frac{x^{n+1}}{n+1}\right]_0^1 = \frac{1}{n+1}$$

abschätzen und bekommen somit $0 \leq I_n \leq \frac{1}{n+1}$ für alle $n > 0$. Damit sehen wir, dass die Vorwärtsrekursion ab $I_{18}$ deutlich falsche Ergebnisse liefert.

Es sei nun $I_n$ der exakte Wert und $\tilde{I}_n$ ein mit Rundungsfehler behafteter Wert. Weiterhin sei $\Delta I_0 = \tilde{I}_0 - I_0$ der Fehler im Startwert, etwa $|\Delta I_0| \approx 10^{-16}$. Dann gilt $I_n = 1 - nI_{n-1}$ und $\tilde{I}_n = 1 - n\tilde{I}_{n-1}$. Daraus folgt $\tilde{I}_n - I_n = -n\left(\tilde{I}_{n-1} - I_{n-1}\right)$, also $\Delta I_n = -n\Delta I_{n-1}$. Sukzessives Einsetzen liefert

$$\Delta I_n = -n\Delta I_{n-1} = -n(-(-n-1))\Delta I_{n-2} = \cdots = (-1)^n n!\Delta I_0$$

und wir erhalten die Fehlerformel $\Delta I_n = (-1)^n n!\Delta I_0$. Dies erklärt das alternierende Vorzeichen für großes $n$. Für $n = 18$ haben wir $18!10^{-16} = 0{,}64023$ und für $n = 19$ haben wir bereits $19!10^{-16} = 12{,}16$. Der Algorithmus ist also instabil.

Eine Alternative zur Berechnung der Integrale ist die Rückwärtsrekursion $I_{n-1} = \frac{1-I_n}{n}$. Das Problem hier: Der Startwert ist unbekannt. Wir können aber $I_n = \frac{1}{n+1}$ benutzen. Die Fehlerformel liefert $\Delta I_0 = \frac{1}{(-1)^n \cdot n!}\Delta I_n$. Auf diese Weise liefert MATLAB den Wert $I_0 \approx 0{,}632120558828555$, wohingegen der „exakte“ Wert $\frac{e-1}{e} = 0{,}632120558828558$ ist. Der Algorithmus ist also stabil.

```
%% Integral-Rekursion:
%% Vorwärtsrekursion
I = zeros(30,1);
I(1) = (exp(1)-1.)/exp(1);
for i=2:30
    I(i) = 1 - (i-1)*I(i-1);
end

%% Rückwärtsrekursion
n0=15;
I = zeros(n0,1);
I(n0) = 1./(n0+1);
for i=n0-1:-1:1
    I(i) = (1.-I(i+1))/(i);
end
```

**MATLAB-Code 4:** Vorwärts- und Rückwärtsrekursion für die Integral-Rekursion in MATLAB.

# 2 Interpolation und Approximation

Häufig tritt in der numerischen Mathematik die Situation auf, dass statt einer Funktion $f : \mathbb{R} \to \mathbb{R}$ nur einige diskrete Funktionswerte $f(x_i)$ und eventuell noch Ableitungen $f^{(j)}(x_i)$ an endlich vielen Punkten $x_i$ gegeben sind. Zum Beispiel, wenn $f$ nur in Form von experimentellen Daten $f(x_0), \ldots, f(x_n)$ vorliegt und man an weiteren Funktionswerten zwischen den tabellierten Werten interessiert ist. Das Ziel ist es also, aus den gegebenen Daten eine Funktion $\varphi$ zu konstruieren, die

$$\varphi^{(j)}(x_i) = f^{(j)}(x_i)$$

für alle $i$ und $j$ erfüllt und sich möglichst wenig von $f$ unterscheidet, also $\|\varphi - f\|$ „klein“ und $\varphi$ leicht auswertbar. Leicht auswertbar sind beispielsweise Funktionen aus $P$, wobei $P$ eine Klasse von einfach strukturierten Funktionen ist, wie etwa

- Polynome $p(x) = a_0 + a_1 x + \cdots + a_n x^n$,
- rationale Funktionen $r(x) = \frac{a_0 + \cdots + a_n x^n}{b_0 + \cdots + b_n x^n}$,
- oder trigonometrische Polynome $t(x) = \frac{1}{2}a_0 + \sum_{k=1}^{b} \{a_k cos(kx) + b_k sin(kx)\}$.

**Definition 2.1.** Geschieht die Zuordnung eines Elements $\varphi \in P$ zur Funktion $f$ durch Fixierung von Funktionswerten

$$\varphi(x_i) = f(x_i), \quad i = 0, \ldots, n,$$

so spricht man von *Interpolation*. Ist $\varphi \in P$ als in einem gewissen Sinne „beste“ Darstellung von $f$ zu bestimmen, zum Beispiel

$$\max_{a \le x \le b} |f(x) - \varphi(x)| \quad \text{minimal für} \quad \varphi \in P$$

oder

$$\left( \int_a^b |f(x) - \varphi(x)|^2 \, \mathrm{d}x \right)^{1/2} \quad \text{minimal für} \quad \varphi \in P,$$

so spricht man allgemein von *Approximation*. Die jeweilige Wahl der Konstruktion von $\varphi \in P$ hängt von der zu erfüllenden Aufgabe ab. Offenbar ist die Interpolation eine spezielle Art der Approximation mit

$$\max_{i=0,\ldots,n} |f(x) - \varphi(x)| \quad \text{minimal für} \quad \varphi \in P.$$

## 2.1 Polynominterpolation

In diesem Abschnitt betrachten wir $P = \mathbb{P}_n$, den Vektorraum der Polynome vom Grad kleiner oder gleich $n$, also

$$\mathbb{P}_n = \{p(x) = a_0 + a_1 x + \cdots + a_n x^n \mid a_i \in \mathbb{R}, i = 0, \ldots, n\}.$$

**Definition 2.2.** Die Polynom-Interpolationsaufgabe besteht darin, zu $n+1$ paarweise verschiedenen Stützstellen (auch Knoten genannt) $x_0, \ldots, x_n \in \mathbb{R}$ und gegebenen Knotenwerten $y_0, \ldots, y_n \in \mathbb{R}$, ein Polynom $p \in \mathbb{P}_n$ mit der Eigenschaft

$$p(x_i) = y_i, \quad i = 0, \ldots, n \qquad \text{(Interpolationsbedingung)}$$

zu bestimmen.

**Satz 2.3.** *Die Polynom-Interpolationsaufgabe ist eindeutig lösbar.*

*Beweis.* Eindeutigkeit: Seien $p, q \in \mathbb{P}_n$ zwei interpolierende Polynome mit

$$p(x_i) = q(x_i), \quad i = 0, \ldots, n.$$

Dann ist $p - q \in \mathbb{P}_n$ ein Polynom mit den $n+1$ Nullstellen $x_0, \ldots, x_n$, da alle Knoten paarweise verschieden sind. Aus dem Satz von Rolle folgt damit sofort, dass $p - q$ das Nullpolynom ist, also $p = q$.

Existenz: Wir wollen ein Polynom $p(x) = a_n x^n + \cdots + a_1 x + a_0$ konstruieren, das die Interpolationsaufgabe erfüllt. Die Koeffizienten erhalten wir als Lösung des linearen Gleichungssystems

$$\underbrace{\begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{pmatrix}}_{=:V_n} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{pmatrix}.$$

Dieses lineare Gleichungssystem hat genau dann eine eindeutige Lösung, wenn $\det(V_n) \neq 0$. Man kann zeigen (siehe Übung), dass

$$\det(V_n) = \prod_{i=0}^{n} \prod_{j=i+1}^{n} (x_j - x_i),$$

also $\det(V_n) \neq 0$ genau dann, wenn die Knoten paarweise verschieden sind. □

*Bemerkung.* Für größere Gleichungssysteme ist die Vandermonde-Matrix $V_n$ im Allgemeinen schlecht konditioniert. Daher ist dieser Ansatz ungeeignet zur Berechnung des Interpolationspolynoms.

### 2.1.1 Darstellung des Interpolationspolynoms

Es seien $x_0 < x_1 < \cdots < x_n$ paarweise verschiedene Knoten.

**Satz 2.4** (Lagrange-Darstellung)**.** *Das (eindeutige) Interpolationspolynom*

$$p_n(x) := p(f|x_0, \ldots, x_n) \in \mathbb{P}_n \quad \textit{mit} \quad p_n(x_j) = f(x_j), \quad j = 0, \ldots, n$$

*lässt sich in der Form*

$$p_n(x) = \sum_{j=0}^{n} f(x_j) l_{jn}(x) \quad \textit{mit} \quad l_{jn}(x) = \prod_{k \neq j}^{n} \frac{x - x_k}{x_j - x_k}, \quad j = 0, \ldots, n$$

*schreiben. Die $l_{jn}(x)$ heißen* Lagrange-Fundamentalpolynome*.*

*Beweis.* Es gilt $l_{jn} \in \mathbb{P}_n$ für alle $j$ und somit $p_n \in \mathbb{P}_n$. Weiterhin ist $l_{jn}(x_i = \delta_{ij}$, wobei $\delta_{ij}$ das Kronecker-Symbol ist. Und schließlich folgt damit

$$p_n(x_i) = \sum_{j=0}^{n} f(x_j) l_{jn}(x_i) = f(x_i).$$

Also löst $p_n$ die Interpolationsaufgabe. Die Eindeutigkeit folgt mit Satz 2.3. □

Der Vorteil dieser Darstellung liegt in der Nützlichkeit für theoretische Betrachtungen. Der Nachteil dieser Darstellung hingegen ist, dass bei Hinzunahme einer neuen Stützstelle die gesamte Darstellung neu berechnet werden muss. Außerdem ist für Stützstellen $x_i \approx x_j$ diese Darstellung sehr ungenau. Dies führt uns zu einer alternativen Darstellung, der Newton-Darstellung.

**Lemma 2.5.** *Für die Lagrange-Interpolationspolynome*

$$p_{n-1} = p(f|x_0, \ldots, x_{n-1}) \in \mathbb{P}_{n-1} \quad \textit{und} \quad p_n = p(f|x_0, \ldots, x_n) \in \mathbb{P}_n$$

*gilt*

$$p_n(x) = p_{n-1}(x) + \delta_n w_n(x)$$

*mit*

$$w_n(x) = \prod_{i=0}^{n-1} (x - x_i) \in \mathbb{P}_n \quad \textit{und} \quad \delta_n = \frac{f(x_n) - P_{n-1}(x_n)}{w_n(x_n)}.$$

*Hierbei nennt man $\{w_i \mid i = 0, \ldots, n\}$ die* Newton-Basis *von $\mathbb{P}_n$ mit*

$$w_0 = 1, w_1 = (x - x_0), w_2 = (x - x_0)(x - x_1), \ldots$$

*Beweis.* Per Definition gilt $p_{n-1} \in \mathbb{P}_{n-1}$ und $w_n \in \mathbb{P}_n$. Damit folgt

$$q_n(x) = p_{n-1}(x) + \delta_n w_n(x) \in \mathbb{P}_n.$$

Es bleibt zu zeigen, dass $q$ die Interpolationsbedingung an den Stützstellen $x_0, \ldots, x_n$ erfüllt. Für $i = 0, \ldots, n-1$ gilt

$$q_n(x_i) = p_{n-1}(x_i) + \delta_n w_n(x_i) = p_{n-1}(x_i) = f(x_i),$$

da $w_n(x_i) = 0$ nach Definition. Es sei nun $i = n$. Dann ist

$$\begin{aligned} q_n(x_n) &= p_{n-1}(x_n) + \delta_n w_n(x_n) \\ &= p_{n-1}(x_n) + \left( \frac{f(x_n) - p_{n-1}(x_n)}{w_n(x_n)} \right) w_n(x_n) \\ &= f(x_n). \end{aligned}$$

Falls $\delta_n = 0$ ist, so folgt bereits $p_{n-1}(x_n) = f(x_n)$ und Satz 2.3 liefert schließlich $q_n = p_n$ und damit die Eindeutigkeit des Interpolationspolynoms von $f$ an den Stützstellen $x_0, \ldots, x_n$. □

Der Koeffizient $\delta_n$ hängt von $f$ und den Stützstellen $x_i$ ab. Daher hat sich auch die Schreibweise $\delta_n = [x_0, \ldots, x_n]f$ oder $f[x_0, \ldots, x_n]$ eingebürgert. Eine wiederholte Anwendung der Argumentation in Lemma 2.5 liefert uns den folgenden Satz.

**Satz 2.6** (Newtonsche Darstellung der Interpolationspolynoms)**.** *Das Interpolationspolynom zu $f$ und Knoten $x_0, \ldots, x_n$ ist gegeben durch*

$$p(f|x_0, \ldots, x_n) = \sum_{i=0}^{n} ([x_0, \ldots, x_i]f) w_i(x) \quad \textit{mit} \quad w_i(x) = \prod_{j=0}^{i-1} (x - x_j),$$

*also*

$$p(f|x_0, \ldots, x_n) = [x_0]f + [x_0, x_1]f w_1(x) + \cdots + [x_0, \ldots, x_n]f w_n(x).$$

**Lemma 2.7** (Aitken)**.** *Es gilt*

$$p(f|x_0, \ldots, x_n)(x) = \frac{x - x_0}{x_n - x_0} p(f|x_1, \ldots, x_n)(x) + \frac{x_n - x}{x_n - x_0} p(f|x_0, \ldots, x_{n-1})(x).$$

*Die Interpolierende an $x_0, \ldots, x_n$ ist also eine lineare Interpolation zwischen zwei Interpolationspolynomen auf $x_1, \ldots, x_n$ und $x_0, \ldots, x_{n-1}$.*

*Beweis.* Wir setzen $x_i$ in die rechte Seite ein und schauen uns die Terme für verschiedene Werte von $i$ getrennt an. Es sei also zunächst $0 < i < n$. Dann gilt:

$$\begin{aligned} &\frac{x_i - x_0}{x_n - x_0} p(f|x_1, \ldots, x_n)(x_i) + \frac{x_n - x_i}{x_n - x_0} p(f|x_0, \ldots, x_{n-1})(x_i) \\ &= \frac{x_i - x_0}{x_n - x_0} f(x_i) + \frac{x_n - x_i}{x_n - x_0} f(x_i) = f(x_i) = p(f|x_0, \ldots, x_n)(x_i). \end{aligned}$$

Für den Fall $i = 0$ fällt der erste Summand weg und wir erhalten

$$\frac{x_n - x_0}{x_n - x_0} p(f|x_0, \ldots, x_{n-1})(x_0) = p(f|x_0, \ldots, x_{n-1})(x_0) = f(x_0).$$

Analog fällt für $i = n$ der zweite Summand weg und es folgt

$$\frac{x_n - x_0}{x_n - x_0} p(f|x_1, \ldots, x_n)(x_0) = p(f|x_1, \ldots, x_n)(x_0) = f(x_0).$$

Also stimmen beide Seiten für alle $x_0, \ldots, x_n$ überein und sind eindeutiges Interpolationspolynom vom Grad kleiner oder gleich $n$. □

*Bemerkung* 2.8. i) Für paarweise verschiedene $x_i$ gilt

$$[x_0, \ldots, x_n]f = \frac{[x_1, \ldots, x_n]f - [x_0, \ldots, x_{n-1}]f}{x_n - x_0}.$$

Diesen Ausdruck nennt man *dividierte Differenz* der Ordnung $n$ von $f$.

ii) Die dividierte Differenz $[x_0, \ldots, x_n]f$ ist der führende Koeffizient $a_n$ des Interpolationspolynoms

$$p(f|x_0, \ldots, x_n)(x) = a_n x^n + \cdots + a_1 x + a_0.$$

*Beweis.* Zu i): Nach Lemma 2.7 gilt

$$p(f|x_0, \ldots, x_n)(x) = \frac{x - x_0}{x_n - x_0} p(f|x_1, \ldots, x_n)(x) + \frac{x_n - x}{x_n - x_0} p(f|x_0, \ldots, x_{n-1})(x).$$

Setze die Newton-Darstellung für das Interpolationspolynom ein und führe Koeffizientenvergleich für den führenden Koeffizienten durch. Die Newton-Darstellung ist

$$p(f|x_0, \ldots, x_n)(x) = \sum_{i=0}^{n} ([x_0, \ldots, x_i]f) w_i(x) \quad \text{mit} \quad w_i(x) = \prod_{j=0}^{i-1} (x - x_j).$$

Koeffizientenvergleich liefert uns jetzt

$$[x_0, \ldots, x_n]f = \frac{1}{x_n - x_0}[x_1, \ldots, x_n]f - \frac{1}{x_n - x_0}[x_0, \ldots, x_{n-1}]f.$$

Zu ii): Folgt aus Lemma 2.5. □

Schema zur Berechnung der dividierten Differenzen für gegebene Startwerte $[x_i]f = f(x_i)$ für $i = 0, \ldots, n$:

$$
\begin{array}{llll}
x_0 & [x_0]f & & \\
 & & [x_0, x_1]f & \\
x_1 & [x_1]f & & [x_0, x_1, x_2]f \\
 & & [x_1, x_2]f & & [x_0, x_1, x_2, x_3]f \\
x_2 & [x_2]f & & [x_1, x_2, x_3]f \\
 & & [x_2, x_3]f & \\
x_3 & [x_3]f & &
\end{array}
$$

**Algorithmus 2.9** (Berechnung dividierter Differenzen).

```
function d = DivDiff(x,f)
    n = size(x,1);

    %Matrix in der die div. Differenzen gespeichert werden.
    d = zeros(n);

    %Schritt 1: Setze erste Spalte.
    d(:,1)=f(x);

    %Schritt 2: Berechne restliche Werte.
    for k = 2:n
        for i = 1:n-k+1
            d(i,k) = (d(i+1,k-1) - d(i,k-1))/(x(i+k-1)-x(i));
        end
    end
end
```

**MATLAB-Code 5:** Berechnung dividierter Differenzen

Beachte: Zur effizienten algorithmischen Beschreibung wird ein doppelter Index benutzt.

**Algorithmus 2.10** (Auswertung der Newton-Interpolationsformel).

```
        %Berechne div. Differenzen.
        D=DivDiff(x,f);

        %Vektor fuer die Loesung.
        Y = zeros(size(X,1),1);

        %Berechnung des Interpolationspolynoms (Alg. 2.10)
        Y(:) = D(1,end);
        for k = size(D,1)-1:-1:1
            Y(:) = D(1,k) + (X(:)-x(k)).*Y(:);
        end
```

**MATLAB-Code 6:** Auswertung der Newton-Interpolationsformel basierend auf Algorithmus 2.9

Aufwand: Es werden $n^2/2$ Divisionen zur Berechnung von $\Delta_{0,i}$ benötigt (wird einmal durchgeführt) und $n$ Multiplikationen zur Berechnung von $y^*$ (wird für jeden Auswertungspunkt $x^*$ durchgeführt) benötigt.

*Bemerkung* 2.11. Für die dividierten Differenzen gilt:

(i) $[x_0, \ldots, x_n]f$ ist unabhängig von der Reihenfolge der $x_i$.

(ii) Ist $f \in C^n(I, \mathbb{R})$, $I = [a, b]$, $a \leq x_0 < x_1 < \cdots < x_n \leq b$, so gibt es ein $\eta \in I$ mit $[x_0, \ldots, x_n]f = \frac{1}{n!} f^{(n)}(\eta)$.

(iii) Ist $p \in \mathbb{P}_{n-1}$, so gilt $[x_0, \ldots, x_n]p = 0$

*Beweis.* Zu i): Folgt aus der Newton-Darstellung des Interpolationspolynoms.

Zu ii): Sei $g := f - p(f|x_0, \ldots, x_n) \in C^n(I, \mathbb{R})$. Dann hat $g$ die $n+1$ Nullstellen $x_0, \ldots, x_n$. Aus dem Satz von Rolle folgt die Existenz eines $\eta \in I$ mit

$$0 = g^{(n)}(\eta) = f^{(n)}(\eta) - p(f|x_0, \ldots x_n)^{(n)}(\eta) = f^{(n)}(\eta) - n![x_0, \ldots, x_n]f.$$

Zu iii): Ist klar, denn der $n$-te Koeffizient eines Polynoms vom Grad kleiner oder gleich $n - 1$ verschwindet. □

### 2.1.2 Verfahrensfehler der Polynominterpolation

Wir stellen uns die Frage, wie stark das Interpolationspolynom von der zu interpolierenden Funktion abweicht. Das Interpolationspolynom erkennt nicht, von welcher Funktion die Daten kommen. Für festes $\overline{x} \in [x_0, x_n]$ gilt aber immer

$$f(\overline{x}) = P(f|x_0, \ldots, x_n, \bar{x})(\overline{x}) \quad \text{mit} \quad P(f|x_0, \ldots, x_n, \overline{x}) \in \mathbb{P}_{n+1}.$$

Daraus folgt mit Lemma 2.5

$$\begin{aligned} f(x) - P(f|x_0, \ldots, x_n)(x) &= P(f|x_0, \ldots, x_n, x)(x) - P(f|x_0, \ldots, x_n)(x) \\ &= [x_0, \ldots, x_n, x]f \cdot w_{n+1}(x) \end{aligned}$$

mit

$$w_{n+1}(x) = \prod_{i=0}^{n} (x - x_i).$$

Wir wollen die linke Seite abschätzen. Nun gibt es nach Bemerkung 2.11 (ii) ein $\xi \in I = [a, b]$, $a \leq x_0$, $b \geq x_n$ mit

$$[x_0, \ldots, x_n, x]f = \frac{f^{(n+1)}(\xi)}{(n+1)!},$$

falls $f \in C^{n+1}(I, \mathbb{R})$. Damit erhalten wir die folgende Abschätzung für den Interpolationsfehler:

**Satz 2.12.** *Sei $f \in C^{n+1}([x_0, x_n], \mathbb{R})$. Dann existiert ein $\xi(x) \in [x_0, x_n]$ mit*

$$f(x) - P(f|x_0, \ldots, x_n)(x) = w_{n+1}(x)\frac{f^{(n+1)}(\xi)}{(n+1)!}.$$

*Insbesondere gilt:*

$$\|f - P(f|x_0, \ldots x_n\|_\infty \leq \|w_{n+1}\|_\infty \frac{\|f^{(n+1)}\|_\infty}{(n+1)!},$$

*wobei $\|f\|_\infty = \max\limits_{x\in[x_0,x_n]} |f(x)|$ ist.*

*Bemerkung.* (i) Der Faktor $\|f^{(n+1)}\|_\infty$ hängt von $f$, aber nicht von den $x_i$ ab,

(ii) und $w_{n+1} = \prod\limits_{j=0}^{n}(x - x_j)$ hängt von den Stützstellen ab, nicht jedoch von $f$.

Eine grobe Abschätzung erhält man durch $|w_{n+1}(x)| \leq (b-a)^{n+1}$ für alle $x \in [a, b]$. Hierbei geht jedoch der Einfluss der Wahl der Sütztstellen verloren.

*Bemerkung.* Wählt man als Stützstellen für die Interpolationsaufgabe auf $[-1, 1]$ die Nullstellen der Tschebyscheff-Polynome, so wird $\|w_{n+1}\|_{\infty,[-1,1]}$ unter allen (normierten) Polynomen mit reellen Nullstellen minimal (Übung).

Die *Tschebyscheff-Polynome* sind eine spezielle Folge von Polynomen $T_k$, die bezüglich eines speziellen gewichteten Skalarprodukts auf $[-1, 1]$ orthogonal sind, das heißt

$$\int_{-1}^{1} \frac{1}{\sqrt{1-x^2}} \cdot T_n(x) \cdot T_m(x)\,\mathrm{d}x = \begin{cases} 0, & m \leq n, \\ \pi, & n = m = 0, \\ \frac{\pi}{2} & n = m \neq 0. \end{cases}$$

Explizit lautet das k-te Polynom

$$T_k(x) = \cos(k \cdot \arccos(x)), \quad x \in [-1, 1].$$

Die Tschebyscheff-Polynome lassen sich auch durch eine Drei-Term-Rekursion berechnen und es gilt

$$T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x), \quad k \geq 2, \quad T_0(x) = 1, \quad T_1(x) = x.$$

Des Weiteren sind die Nullstellen von $T_k(x)$ gegeben durch

$$x_j = \cos\left(\frac{2j+1}{2k}\pi\right), \quad j = 0, \ldots, k-1.$$

### Grenzen der Polynominterpolation

Durch Erhöhung der Anzahl der Stützstellen $n$ kann man nicht automatisch eine beliebig genaue Approximation an $f$ bezüglich der $\|\cdot\|_\infty$-Norm erreichen. Runge hat hierfür ein Gegenbeispiel angegeben. Betrachte

$$f(x) = \frac{1}{1+x^2} \in C^\infty, \quad x \in [-c, c], \quad c > 0$$

mit äquidistanten Stützstellen

$$x_i = -c + \frac{2c}{n} i, \quad i = 0, \ldots, n.$$

Es gilt:

- $\|f - P_n\|_\infty \to \infty$ für $n \to \infty$, falls $c > \frac{e}{2}$,
- $\|f - P_n\|_\infty \to \infty$ für $n \to \infty$, falls $c \leq \frac{e}{2}$.

Im divergenten Fall treten insbesondere an den Intervallrändern immer stärkere Oszillationen auf. Die bezeichnet man auch als das Runge-Phänomen.

*Bemerkung*. Der Weierstraßsche Approximationssatz besagt, dass jede Funktion $f \in C([a, b])$ beliebig gut gleichmäßig auf $[a, b]$ durch Polynome approximiert werden kann. Die Vermutung, dass dies mit Interpolationspolynomen geschehen kann, ist jedoch im Allgemeinen falsch.

```
% Interpolation
% Runge Testproblem

c_vec = [exp(1)/2+3 exp(1)/2-.5];    % c.
n_vec = [3 10 20 40];                  % n.

f = @(x) 1./(1+x.^2);                %Funktion.

for n = n_vec
p=1;                                  %Subplot-Nummer.
figure;
for c = c_vec
    a = -c;
    b = c;
    X=[-c:2*c/100:c]';                % X fuer den Plot.

    for j = 1:2
        if j==1
            % Aequidistant
            TITLE = [ 'Aequidistant', ' mit c = ',num2str(c), ', n = '...
```

```
                ,num2str(n)];
            x=[-c:2*c/n:c]';
        else
            % Tschebyscheff
            TITLE = ['Tschebyscheff', ' mit c = ',num2str(c), ', n = '...
                ,num2str(n)];
            x=zeros(n,1);
            for i=1:n
                xi = cos(pi*(2*i-1)/(2*n));
                x(i) = a + (b - a)/2*(xi+1);
            end
        end

        %Berechne div. Differenzen.
        D=DivDiff(x,f);

        %Vektor fuer die Loesung.
        Y = zeros(size(X,1),1);

        %Berechnung des Interpolationspolynoms (Alg. 2.10)
        Y(:) = D(1,end);
        for k = size(D,1)-1:-1:1
            Y(:) = D(1,k) + (X(:)-x(k)).*Y(:);
        end

        %Plot
        subplot(2,2,p)
        plot(X,f(X),'-k',X,Y,'-.')
        title(TITLE)
        legend('f(f)','P(x)');
        p=p+1;
    end
end
end
```

**MATLAB-Code 7:** Runge-Phänomen

### 2.1.3 Die baryzentrische Darstellung des Interpolationspolynoms

Um das Problem des Runge-Phänomens zu umgehen, kann man noch eine andere Darstellung des Interpolationspolynoms betrachten, die baryzentrische Darstellung. Wir haben zuvor die Lagrange-Darstellung

$$p(x) = \sum_{j=0}^{n} l_{jn}(x) f_j \quad \text{mit} \quad l_{jn}(x) = \prod_{k \neq j}^{n} \frac{(x - x_k)}{(x_j - x_k)}$$

betrachtet, wobei $l_{jn}(x_k) = \delta_{jk}$ gilt.

Vor- und Nachteile:

+ Ideal, um etwas zu beweisen.
- Auswertung von $p(x)$ benötigt $\mathcal{O}(n^2)$ Rechenoperationen.
- Hinzufügen von $(x_{n+1}, f_{n+1})$ ändert alles.
- Numerisch instabil.

Weiterhin haben wir auch die Newton-Darstellung

$$p(x) = \sum_{j=0}^{n} [x_0, \ldots, x_j] f \prod_{i=0}^{j-1} (x - x_i)$$

mit

$$[x_j, \ldots, x_k]f = \frac{[x_{j+1}, \ldots, x_k]f - [x_j, \ldots, x_{k-1}]f}{x_k - x_j} \quad \text{und} \quad [x_j]f = f_j, \quad j = 0, \ldots, n$$

kennengelernt.

Vor- und Nachteile:

+ Auswertung von $p(x)$ benötigt $\mathcal{O}(n)$ Rechenoperationen.
+ Hinzufügen von $(x_{n+1}, f_{n+1})$ ist leicht umsetzbar.
- Rekursive Formel für $[x_0, \ldots, x_j]f$ benötigt $\mathcal{O}(n^2)$ Operationen.
- $[x_0, \ldots, x_j]f$ ist abhängig von der Funktion $f$.

**Modifizierte Lagrange-Darstellung**

Es sei nun

$$l(x) := (x - x_0) \cdot \cdots \cdot (x - x_n) \quad \text{mit Stützkoeffizienten} \quad \lambda_j = \prod_{k \neq j}^{n} \frac{1}{(x_j - x_k)} = \frac{1}{l'(x_j)}.$$

Jetzt können wir die Lagrange-Basispolynome darstellen durch

$$l_{jn}(x) = \prod_{k \neq j}^{n} \frac{(x - x_k)}{(x_j - x_k)} = l(x) \frac{\lambda_j}{x - x_j}$$

und erhalten für das Interpolatinspolynom

$$p(x) = l(x) \sum_{j=0}^{n} \frac{\lambda_j}{x - x_j} f_j.$$

Diese Darstellung hat die folgenden Eigenschaften:

- Einmalig $\mathcal{O}(n^2)$ Rechenoperationen zur Berechnung von $\lambda_j$ (unabhängig von $f_j$).
- $\mathcal{O}(n)$ Rechenoperationen zur Berechnung von $p(x)$.
- Hinzufügen von $(x_{n+1}, f_{n+1})$:
  - Dividiere $\lambda_j$ durch $x_j - x_{n+1} \to n+1$ Operationen.
  - Berechne $\lambda_{n+1} \to n+1$ Operationen.

Mit Hilfe der $\lambda_j$ und $f(x) \equiv 1$ lässt sich dann $l(x)$ berechnen durch

$$1 = \sum_{j=0}^{n} l_{jn}(x) = l(x) \sum_{j=0}^{n} \frac{\lambda_j}{x - x_j},$$

also

$$l(x) = \frac{1}{\sum_{j=0}^{n} \frac{\lambda_j}{x - x_j}}.$$

Wir haben also gezeigt:

**Satz 2.13** (Baryzentrische Interpolationsformel)**.** *Das Interpolationspolynom $p \in \mathbb{P}_n$ zu den $(n+1)$ Knoten $x_0, \ldots, x_n$ und den $(n+1$ Daten $f_0, \ldots, f_n$ hat die Form*

$$p(x) = \begin{cases} \sum_{j=0}^{n} \frac{\lambda_j f_j}{x - x_j} \Big/ \sum_{j=0}^{n} \frac{\lambda_j}{x - x_j}, & x \neq x_j, \\ f_j, & x = x_j. \end{cases}$$

*Die Stützkoeffizienten haben die Form*

$$\lambda_j = \prod_{k \neq j}^{n} \frac{1}{x_j - x_k}.$$

**Baryzentrische Interpolation in Tschebyscheff-Knoten**

Die Tschebyscheff-Knoten 1. Art sind definiert durch die Nullstellen der Tschebyscheff-Polynome

$$x_j = \cos\left(\frac{(2j+1)\pi}{2n+2}\right), \quad 0 \leq j \leq n$$

Dann erhalten wir für die Gewichte unter Vernachlässigung aller von $j$ unabhängigen Bestandteile

$$\lambda_j = (-1)^j \sin\left(\frac{(2j+1)\pi}{2n+2}\right).$$

In praktischen Rechnungen benutzt man meist die Tschebyscheff-Knoten 2. Art, die gegeben sind durch

$$x_j = \cos(j\pi/n), \quad 0 \le j \le n.$$

In diesem Fall erhält man die Gewichte

$$\lambda_j = (-1)^j \delta_j, \quad \text{mit} \quad \delta_j = \begin{cases} \frac{1}{2}, & j = 0, n, \\ 1, & \text{sonst.} \end{cases}$$

Das Interpolationspolynom hat dann die Form

$$p(x) = \sum_{j=0}^{n}{}' \frac{(-1)^j f_j}{x - x_j} \Big/ \sum_{j=0}^{n}{}' \frac{(-1)^j}{x - x_j}, \quad \text{und} \quad p(x_j) = f_j, \quad j = 0, \ldots, n.$$

Dabei bedeutet $\sum'$, dass der erste und letzte Summand jeweils mit $1/2$ multipliziert werden. Im Gegensatz zu den vorherigen Darstellungen des Interpolationspolynoms, liefert die baryzentrische Darstellung einen stabilen Algorithmus zur Berechnung des Interpolationspolynoms.

## 2.2 Hermite-Interpolation

**Definition 2.14.** Die Hermite-Interpolationsaufgabe lautet:

| | | | |
|---|---|---|---|
| gegeben: | Knoten | $x_0 < x_1 < \cdots < x_m$, | $m \ge 0$, |
| | Werte | $y_i^{(k)} \in \mathbb{R}$, | $i = 0, \ldots, m, k = 0, \ldots, \mu_i$, |
| | | | $\mu_i \ge 0$ für $0 \le i \le m$, |
| gesucht: | $p \in \mathbb{P}_n, n = m + \sum_{i=0}^{m} \mu_i$, | mit $p(x_i)^k = y_i^{(k)}$, | $i = 0, \ldots, m, k = 0, \ldots, \mu_i$. |

**Satz 2.15.** *Die Hermite-Interpolationsaufgabe besitzt eine eindeutige Lösung.*

*Beweis.* Der Beweis wird analog zu dem von Satz 2.3 geführt (Übung). □

Die Hermite-Interpolation ist eine Verallgemeinerung der klassischen Polynom-Interpolation, bei der man die Interpolation von Ableitungen formal als zusammenfallende Stützstellen interpretiert, das heißt $a \le x_0 \le x_1 \le \cdots \le x_n \le b$. Sind in einem Knoten $x_i$ die Funktionswerte $f(x_i)$ und die Ableitungen $f'(x_i), \ldots, f^{(\mu_i)}(x_i)$ bis zum Grad $\mu_i$ gegeben, so soll $x_i$ genau $(\mu_i + 1)$-mal auftreten. Sind alle Knoten paarweise verschieden, so erhalten wir die klassische Polynom-Interpolation. Stimmen alle

Knoten überein, das heißt $x_0 = x_1 = \cdots = x_n$, so ist das Interpolationspolynom die abgebrochene Taylor-Reihe

$$p(f, x_0, \ldots, x_n)(x) = \sum_{j=0}^{n} \frac{(x - x_0)^j}{j!} f^{(j)}(x_0).$$

um $x_0$. Wir können das Interpolationspolynom wieder in der Newton-Darstellung

$$p = \sum_{i=0}^{n} [x_0, \ldots, x_i] f \cdot w_i, wi = \prod_{j=0}^{i-1} (x - x_j)$$

schreiben. Beachte: Die bisherige Definition der dividierten Differenzen können wir nicht ohne Weiteres verwenden, da Stützstellen doppelt auftreten können. Die Kombination von klassischer Interpolation und Taylor-Reihe ergibt

$$\begin{aligned} [x_i, \ldots, x_{i+k}] f &= \frac{f^{(k)}(x_i)}{k!}, && \text{falls} \quad x_i = x_{i+k}, \\ [x_i, \ldots, x_{i+k}] f &= \frac{[x_{i+1}, \ldots, x_{i+k}] f - [x_i, \ldots, x_{i+k-1}] f}{x_{i+k} - x_i} && \text{sonst.} \end{aligned}$$

*Beispiel.* Finde das Hermite-Interpolationspolynom $p \in \mathbb{P}_4$ mit

$$\begin{aligned} &p(0) = f(0) = -1, && p'(0) = f'(0) = -2, \\ &p(1) = f(1) = 0, && p'(1) = f'(1) = 10, && p''(1) = f''(1) = 40. \end{aligned}$$

Wir verdoppeln den ersten und verdreifachen den zweiten Knoten und erhalten die dividierten Differenzen

$$\begin{array}{lllll} x_0 = 0 & [x_0]f = -1 & & & \\ & & [x_0, x_1]f = -2 & & \\ x_1 = 0 & [x_1]f = -1 & & [x_0, x_1, x_2]f = 3 & \\ & & [x_1, x_2]f = 1 & & [x_0, \ldots, x_3]f = 6 \\ x_2 = 1 & [x_2]f = 0 & & [x_1, x_2, x_3]f = 9 & [x_0, \ldots, x_4]f = 5 \\ & & [x_2, x_3]f = 10 & & [x_1, \ldots, x_4]f = 11 \\ x_3 = 1 & [x_3]f = 0 & & [x_2, x_3, x_4]f = 20 & \\ & & [x_3, x_4]f = 10 & & \\ x_4 = 1 & [x_4]f = 0 & & & \end{array}$$

Daraus ergibt sich das Hermite-Interpolationspolynom als

$$\begin{aligned} P(f|x_0, \ldots, x_4) &= -1 - 2(x - x_0) + 3(x - x_0)(x - x_1) \\ &\quad + 6(x - x_0)(x - x_1)(x - x_2) + 5(x - x_0)(x - x_1)(x - x_2)(x - x_3) \\ &= -1 - 2x + 3x^2 + 6x^3(x - 1) + 5x^4(x - 1)^2. \end{aligned}$$

**Satz 2.16** (Fehler der Hermite-Interpolation). *Ist $f \in C^{n+1}([x_0, x_m])$, so gibt es zu jedem $x \in [x_0, x_m]$ ein $\xi(x) \in [x_0, x_m]$, so dass für die Lösung $p \in \mathbb{P}_n$ der Hermite-Interpolationsaufgabe*

$$\begin{aligned} f(x) - p(x) &= [x_0, \dots, x_0, \dots, x_m, \dots x_m, x] f \prod_{i=0}^{m} (x - x_j)^{\mu_i + 1} \\ &= \frac{1}{(n+1)!} f^{(n+1)}(\xi(x)) \prod_{i=0}^{m} (x - x_j)^{\mu_i + 1} \end{aligned}$$

*gilt.*

*Beweis.* Der Beweis funktioniert wie in Satz 2.12. □

## 2.3 Spline-Interpolation

Im Folgenden sei $I = [a, b]$ ein Intervall mit $a < b$. Weiter sei $X$ eine Zerlegung von $I$, das heißt $X : a = x_0 < x_1 < \cdots < x_n = b$, $n \in \mathbb{N}$. Es sei $f_j \in \mathbb{R}$ für $j = 0, \dots, n$.

**Definition 2.17.** Es sei $k \in \mathbb{N}$. Dann heißt

$$S_k(X) = \left\{ S \in C^{k-1}(I) \mid 1 \leq j \leq n, s|_{[x_{j-1}, x_j]} \in \mathbb{P}_k \right\}$$

*Raum der polynomialen Spline-Funktionen* oder *Splines* vom Grad $k$ auf der Zerlegung $X$.

*Beispiel.* (i) Für $k = 1$ erhält man stetige Polygonzüge. Die Interpolationsaufgabe lautet: Finde $s \in S_1(X)$ mit $s(x_j) = f_j$ für $j = 0, \dots, n$.

(ii) Für $k = 2$ erhält man quadratische Splines (wenig genutzt), also stückweise Polynome 2. Grades, die global einmal stetig differenzierbar sind. Beachte: Die Interpolationsaufgabe $s(x_j) = f_j$ für $j = 0, \dots, n$ enthält eine Interpolationsbedingung zu wenig. Man kann zum Beispiel $s'(x_0) = f'(x_0)$ oder $s'(x_n) = f'(x_n)$ zusätzlich angeben.

(iii) Für $k = 3$ erhält man kubische Splines (häufig genutzt), also stückweise Polynome 3. Grades, die global zweimal stetig differenzierbar sind. Beachte: Hier enthält die Interpolationsaufgabe zwei Bedingungen zu wenig. Wähle zum Beispiel $s'(x_0) = f'(x_0)$ und $s'(x_n) = f'(x_n)$ als zusätzliche Interpolationsbedingungen.

### Dimensionsbetrachtungen

Gesucht ist $s \in S_3(X)$ mit $s(x_j) = f_j$ für $j = 0, \dots n$. Da wir Polynome 3. Grades betrachten, haben wir in jedem Teilintervall $[x_i, x_{i+1}]$ von $I$ genau $4$ Parameter. Dies

liefert uns insgesamt $4n$ Parameter für das gesamte Intervall $I$. Durch die Stetigkeitsbedingungen $s^{(i)}(x_j - 0) = s^{(i)}(x_j + 0)$ für $i = 0, 1, 2$ und $j = 1, \ldots, n-1$ haben wir bereits $3(n-1)$ Bedingungen gegeben. Durch die Interpolationsbedingungen $s(x_j) = f_j$ für $j = 0, \ldots, n$ erhalten wir weitere $n+1$ Bedingungen. Wir können also noch

$$4n - 3(n-1) - (n+1) = 2$$

weitere Bedingungen stellen.

Woher kommt das Interesse an diesen Funktionen? Ingenieure benutzten früher Splines, das heißt „dünne Holzlatten" bei der Konstruktion (zum Beispiel im Schiffsbau). Man zwang Splines durch bestimmte Knotenpunkte. Dadurch stellte sich für Rumpflinien von Schiffen eine günstige Kurve ein. Die Holzlatte nimmt eine Lage mit minimaler potentieller Energie an, das heißt

$$\int_a^b \frac{y''(x)^2}{(1 + y'(x)^2)^{3/2}} \, \mathrm{d}x$$

wird minimal. Falls $|y'(x)|$ klein ist, erhalten wir näherungsweise

$$\int_a^b y''(x)^2 \, \mathrm{d}x$$

minimal. Außerhalb von $[a, b]$ verläuft die Latte linear, das heißt wegen $y \in C^2(\mathbb{R})$ gilt $y''(x) = 0$ in $(-\infty, a]$ und $[b, \infty)$. Insbesondere gilt $y''(x_0) = y''(x_n)$ (natürliche Randbedingung). Daraus ergibt sich die

**Aufgabenstellung 2.18.** *Es seien die Zerlegung $X$ und Daten $f_0, \ldots, f_n$ gegeben und es sei*

$$A = \{f \in C^2(I) \mid f(x_j) = f_j,\ j = 0, \ldots, n\}.$$

*Gesucht ist $\tilde{f} \in A$ mit $E(\tilde{f}) \leq E(f)$ für alle $f \in A$. Dabei sei*

$$E(f) := \int_a^b (f''(x))^2 \, \mathrm{d}x$$

*(dies ist nur eine Halbnorm, denn $|f| = 0 \Leftrightarrow f = 0$ ist nicht erfüllt).*

### 2.3.1 Berechnung kubischer Splines

**Satz 2.19.** *Es existiert eine Lösung $\tilde{f} \in A$ von Problem 2.18. Für dieses $\tilde{f}$ gilt $\tilde{f} \in S_3(X)$ und $f''(x_0) = f''(x_n) = 0$. Diese Lösung ist eindeutig.*

*Beweis.* Zur Minimaltiät: Es seien $f, s \in A$. Dabei sei $s \in S_3(X)$ ein natürlicher kubischer Spline (das heißt ein kubischer Spline mit $s''(x_0) = s''(x_n) = 0$). Wir müssen zeigen, dass $E(s) \leq E(f)$ gilt. Es gilt

$$\begin{aligned} 0 \leq E(f-s) &= \int_a^b (f''(x) - s''(x))^2 \,\mathrm{d}x \\ &= \int_a^b (f''(x))^2 \,\mathrm{d}x - 2\int_a^b (f''(x)s''(x)) \,\mathrm{d}x + \int_a^b (s''(x))^2 \,\mathrm{d}x \\ &= \int_a^b (f''(x))^2 \,\mathrm{d}x - 2\int_a^b (f''(x) - s''(x))s''(x) \,\mathrm{d}x - \int_a^b (s''(x))^2 \,\mathrm{d}x. \end{aligned}$$

Wir zeigen nun, dass der zweite Term verschwindet, womit $E(s) \leq E(f)$ gezeigt wäre. Definiere $I_j = (x_{j-1}, x_j)$. Dann gilt

$$\int_a^b (f''(x) - s''(x))s''(x) \,\mathrm{d}x = \sum_{j=1}^n \int_{I_j} (f''(x) - s''(x))s''(x) \,\mathrm{d}x.$$

Wegen $s \in C^\infty(I_j)$ erhalten wir mit partieller Integration und der Tatsache, dass $s'''$ konstant ist

$$\begin{aligned} \int_{I_j} (f'' - s'')s'' \,\mathrm{d}x &= \big[(f' - s')s''\big]_{x_{j-1}}^{x_j} - \int_{I_j} (f' - s')s''' \,\mathrm{d}x \\ &= \big[(f' - s')s''\big]_{x_{j-1}}^{x_j} - s'''\big((f-s)(x_j) - (f-s)(x_{j-1})\big) \\ &= \big[(f' - s')s''\big]_{x_{j-1}}^{x_j}, \end{aligned}$$

da der zweite Term aufgrund der Interpolationsbedingungen verschwindet. Folglich erhalten wir

$$\int_a^b (f'' - s'')s'' \,\mathrm{d}x = \sum_{j=1}^n \big[(f' - s')s''\big]_{x_{j-1}}^{x_j} = \big[(f' - s')s''\big]_{x_0}^{x_n} = 0,$$

da $s$ stetig auf $I$ ist und wegen $s''(x_0) = s''(x_n) = 0$. Also gilt

$$0 \leq \int_a^b (f'' - s'')^2 \,\mathrm{d}x = \int_a^b (f'')^2 \,\mathrm{d}x - \int_a^b (s'')^2 \,\mathrm{d}x$$

für alle $f \in A$. Das heißt aber gerade, dass $s$ die Lösung von Problem 2.18 ist, denn $E(s) \leq E(f)$. Wir haben also bisher gezeigt, dass wenn es eine Lösung von Problem 2.18 gibt, diese ein natürlicher kubischer Spline ist.

Zur Eindeutigkeit: Es seien $s, \tilde{s} \in S_3(X) \cap A$ Lösungen von Problem 2.18. Dann gilt: sowohl $E(s) \leq E(\tilde{s})$ (nach obiger Rechnung), als auch $E(\tilde{s}) \leq E(s)$, also $E(\tilde{s}) = E(s)$. Dann folgt mit obiger Rechnung sowohl $E(s) \leq E(\tilde{s})$ als auch $E(\tilde{s}) \leq E(s)$ und damit $E(\tilde{s}) = E(s)$. Außerdem liefert die obige Rechnung

$$E(\tilde{s} - s) = \int_a^b (\tilde{s}'' - s'')^2 \,\mathrm{d}x = \int_a^b (\tilde{s}'')^2 \,\mathrm{d}x - \int_a^b (s'')^2 \,\mathrm{d}x = E(\tilde{s}) - E(s) = 0.$$

Also gilt $\tilde{s}''(x) - s''(x) = 0$ für alle $x \in [a, b]$. Mit Integration folgt nun $\tilde{s}(x) = s(x) + cx + d$ mit $c, d \in \mathbb{R}$. Wegen der Interpolationsbedingung sind $s$ und $\tilde{s}$ mindestens an $x_0$ und $x_n$ gleich. Daraus folgt wegen

$$\begin{aligned} \tilde{s}(x_0) &= s(x_0) + cx_0 + d, \\ \tilde{s}(x_n) &= s(x_n) + cx_n + d, \end{aligned}$$

dass $cx_0 + d = 0 = cx_n + d$, woraus wiederum $cx_0 = cx_n$ folgt und wegen $x_0 \neq x_n$ schließlich $c = 0$ und damit auch $d = 0$, also $s = \tilde{s}$.

Zur Existenz (konstruktiv): Wir wollen schließlich noch zeigen, dass es einen natürlichen kubischen Spline gibt. Im folgenden werden wir aber die Konstruktion eines solchen Splines etwas allgemeiner darstellen und auch andere Interpolationsprobleme lösen.

Es sei $s \in S_3(X)$ und $s''$ ist in jedem Teilintervall ein Polynom 1. Grades und global ein stetiger Polygonzug, also

$$s''(x)\Big|_{I_j} = a_j x + b_j, \quad a_j, b_j \in \mathbb{R}, \quad j = 1, \ldots, n.$$

Setze $M_j := s''(x_j)$ für $j = 0, \ldots, n$ und $h_j = x_j - x_{j-1}$ für $j = 1, \ldots, n$. Dann folgt

$$s''(x)\Big|_{I_j} = \frac{1}{h_j}\big(M_{j-1}(x_j - x) + M_j(x - x_{j-1})\big),$$

wegen der Stetigkeit von $s''$. Es gilt also

$$\begin{aligned} s'(x) &= \frac{1}{h_j}\left(M_j \frac{(x - x_{j-1})^2}{2} - M_{j-1}\frac{(x_j - x)^2}{2}\right) + c_j && \text{in } I_j, \\ s(x) &= \frac{1}{h_j}\left(M_j \frac{(x - x_{j-1})^3}{6} + M_{j-1}\frac{(x_j - x)^3}{6}\right) + c_j\,(x - x_{j-1}) + d_j && \text{in } I_j. \end{aligned}$$

Die Stetigkeit der ersten Ableitung in $x_j$ für $j = 1, \ldots, n-1$ liefert $s'(x_j - 0) = s'(x_j + 0)$, woraus

$$\frac{h_j}{2} M_j + c_j = -M_j \frac{h_{j+1}}{2} + c_{j+1} \qquad (*)$$

folgt und die Interpolationsbedingungen $s(x_j) = f(x_j)$ für $j = 0, \ldots, n$ ergeben

$$f_{j-1} = \frac{h_j^2}{6} M_{j-1} + d_j \qquad \text{in } I_j,$$
$$f_j = \frac{h_j^2}{6} M_j + c_j h_j + d_j \qquad \text{in } I_j$$

und damit

$$\frac{f_j - f_{j-1}}{h_j} = \frac{h_j}{6}(M_j - M_{j-1}) + c_j. \tag{$**$}$$

Folglich gilt

$$c_j = [x_{j-1}, x_j]\, f - \frac{h_j}{6}(M_{j-1}), \quad d_j = f_{j-1} - \frac{h_j^2}{6} M_{j-1}, \quad j = 1, \ldots, n.$$

Setze nun $c_j$ in $(*)$ ein und erhalte

$$h_j M_{j-1} + 2(h_j + h_{j+1}) M_j + h_{j+1} M_{j+1} = 6\left([x_j, x_{j+1}]\, f - [x_{j-1}, x_j]\, f\right), \quad j = 1, \ldots, n-1.$$

Dies sind $n-1$ lineare Gleichungen für $n+1$ Unbekannte $M_0, \ldots, M_n$. Dieses Gleichungssystem wird in praktischen Rechnungen benutzt. Für theoretische Betrachtungen dividieren wir noch durch $h_j + h_{j+1} = x_{j+1} - x_{j-1}$ und erhalten

$$\mu_j M_{j-1} + 2M_j + \lambda_j M_{j+1} = 6\,[x_{j-1}, x_j, x_{j+1}]\, f \tag{GS}$$

mit

$$\mu_j = \frac{h_j}{h_j + h_{j+1}}, \quad \lambda_j = \frac{h_{j+1}}{h_j + h_{j+1}}, \quad j = 1, \ldots, n-1,$$

wobei $\mu_j, \lambda_j > 0$ und $\mu_j + \lambda_j = 1$ gilt. □

### Möglichkeiten zur Bestimmung der $M_i$

**1. Fall:** $M_0 = M_n = 0$ **(natürliche Splines)**

Das $(n-1)\times(n-1)$-Gleichungssystem mit den Unbekannten $M_1, \ldots, M_{n-1}$ ist gegeben durch

$$\begin{pmatrix} 2 & \lambda_1 & & & \\ \mu_2 & 2 & \lambda_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \mu_{n-2} & 2 & \lambda_{n-2} \\ & & & \mu_{n-1} & 2 \end{pmatrix} \begin{pmatrix} M_1 \\ \vdots \\ M_{n-1} \end{pmatrix} = 6 \begin{pmatrix} [x_0, x_1, x_2]\, f \\ \vdots \\ [x_{n-2}, x_{n-1}, x_n]\, f \end{pmatrix}$$

**2. Fall: Zusätzliche Interpolationsbedingungen (vollständige Randbedingungen)**

Wir fordern zusätzlich $s'(x_0) = f'(x_0)$ und $s'(x_n) = f'(x_n)$. Für $j = 1$ folgt aus der Interpolationsbedingung $(**)$

$$[x_0, x_1]\, f = \frac{f_1 - f_0}{h_1} = \frac{h_1}{6}(M_1 - M_0) + c_1$$

und aus

$$s'(x) = \frac{1}{h_j}\left[M_j \frac{(x - x_j)^2}{2} - M_{j-1}\frac{(x_j - x)^2}{2}\right] + c_j \quad \text{in} \quad I_j$$

folgt für $x = x_0$ und $j = 1$ die Gleichung

$$s'(x_0) = f'(x_0) = [x_0, x_0]\, f = \frac{1}{h_1}\left[-M_0 \frac{h_1^2}{2}\right] + c_1 = -M_0 \frac{h_1}{2} + c_1.$$

Damit erhalten wir

$$[x_0, x_0, x_1]\, f = \frac{[x_0, x_1]\, f - f'(x_0)}{x_1 - x_0} = \frac{1}{6}(M_1 - M_0) + \frac{c_1}{h_1} + \frac{M_0}{2} - \frac{c_1}{h_1} = \frac{1}{6}(M_1 + 2M_0).$$

Eine analoge Gleichung wird aus der Bedingung $s'(x_n) = f'(x_n)$ hergeleitet. Zusätzlich zu (GS) erhalten wir somit

$$\begin{aligned} 2M_0 + M_1 &= 6\,[x_0, x_0, x_1]\, f, \\ M_{n-1} + 2M_n &= 6\,[x_{n-1}, x_n, x_n]\, f, \end{aligned}$$

also das $(n+1) \times (n+1)$-Gleichungssystem

$$\begin{pmatrix} 2 & 1 & & & \\ \mu_2 & 2 & \lambda_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \mu_{n-1} & 2 & \lambda_{n-2} \\ & & & 1 & 2 \end{pmatrix} \begin{pmatrix} M_1 \\ \\ \vdots \\ \\ M_{n-1} \end{pmatrix} = 6 \begin{pmatrix} \delta_0 \\ \\ \vdots \\ \\ \delta_n \end{pmatrix}$$

mit

$$\delta_0 = [x_0, x_0, x_1]\, f, \quad \delta_n = [x_{n-1}, x_n, x_n]\, f, \quad \delta_j = [x_{j-1}, x_j, x_{j+1}]\, f, \quad j = 1, \ldots, n-1.$$

**3. Fall: Periodische Daten** $f_n = f_0$

In diesem Fall setzen wir das Gitter mit der Periode $(b - a)$ periodisch auf $\mathbb{R}$ fort. Wir erhalten dadurch aus der Periodizität von $s \in C^2$ zwei zusätzliche Gleichungen

$s'(x_0) = s'(x_n)$ und $s''(x_0) = s''(x_n)$. Es folgt $M_0 = M_n$ und damit verbleiben $n$ Unbekannte $M_1, \ldots, M_{n-1}, M_n = M_0$. Folglich erhalten wir eine zusätzliche Bestimmungsgleichung

$$\mu_n M_{n-1} + 2M_n + \lambda_n M_{n+1} = 6\tilde{\delta}_n,$$

wobei $\tilde{\delta}_n = [x_{n-1}, x_n, x_{n+1}]\, f$ und $x_{n+1} = x_1 + (b - a)$. Das Gleichungssystem lautet dann:

$$\begin{pmatrix} 2 & \lambda_1 & & & \mu_1 \\ \mu_2 & 2 & \lambda_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \mu_{n-1} & 2 & \lambda_{n-2} \\ \lambda_n & & & \mu_n & 2 \end{pmatrix} \begin{pmatrix} M_1 \\ \\ \vdots \\ \\ M_n \end{pmatrix} = 6 \begin{pmatrix} \delta_1 \\ \\ \vdots \\ \\ \tilde{\delta}_n \end{pmatrix}$$

mit $\delta_i$ wie in Fall 2.

**4. Fall: not-a-knot-Spline**

Als zusätzliche Bedingungen setzen wir $s'''(x_1 - 0) = s'''(x_1 + 0)$ und $s'''(x_{n-1} + 0) = s'''(x_{n+1} + 0)$, das heißt $s$ ist auf $[x_0, x_2]$ und $[x_{n-2}, x_n]$ ein Polynom 3. Grades, $x_1$ und $x_{n-1}$ sind also keine Knoten im eigentlichen Sinne. Es gilt

$$s'''(x) = \frac{1}{h_j} = \frac{1}{h_j}(M_j - M_{j-1})$$

und damit

$$s'''(x) = \frac{1}{h_1}(M_1 - M_0) \quad \text{in } I_1,$$
$$s'''(x) = \frac{1}{h_2}(M_2 - M_1) \quad \text{in } I_2.$$

Die erste zusätzliche Bedingung liefert uns

$$\frac{1}{h_1}(M_1 - M_0) = \frac{1}{h_2}(M_2 - M_1),$$

woraus wir

$$M_0\lambda_1 - M_1 + M_2\mu_1 = 0$$

erhalten. Analog ergibt sich für $I_{n-1}$ und $I_n$ mit der zweiten zusätzlichen Bedingung

$$M_{n-2}\lambda_{n-1} - M_{n-1} + M_n\mu_{n-1} = 0.$$

Insgesamt ist das Gleichungssystem dann durch

$$\begin{pmatrix} \lambda_1 & -1 & \mu_1 & & & & \\ \mu_1 & 2 & \lambda_1 & & & & \\ & \mu_2 & 2 & \lambda_2 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & \mu_{n-2} & 2 & \lambda_{n-2} & \\ & & & & \mu_{n-1} & 2 & \lambda_{n-1} \\ & & & & \lambda_{n-1} & -1 & \mu_{n-1} \end{pmatrix} \begin{pmatrix} M_1 \\ \\ \\ \vdots \\ \\ \\ M_n \end{pmatrix} = 6 \begin{pmatrix} 0 \\ \delta_1 \\ \\ \vdots \\ \\ \delta_n \\ 0 \end{pmatrix}$$

gegeben, wobei auch hier $\delta_i$ wie in den Fällen zuvor definiert ist.

Nun ergibt sich die Frage, ob diese linearen Gleichungssysteme lösbar sind.

**Definition 2.20.** Eine Matrix $A \in \mathbb{R}^{nxn}$ (oder $A \in \mathbb{C}^{nxn}$) heißt *(strikt) diagonaldominant*, falls

$$|a_{ii}| > \sum_{k=1, k\neq i}^{n} |a_{ik}|, \quad i = 1, \ldots, n.$$

Sie heißt *schwach diagonaldominant*, wenn stattdessen

$$|a_{ii}| \geq \sum_{k=1, k\neq i}^{n} |a_{ik}|, \quad i = 1, \ldots, n$$

und für mindestens einen Index die strikte Ungleichung gilt.

**Lemma 2.21.** *Es sei $A \in \mathbb{R}^{nxn}$ (oder $A \in \mathbb{C}^{nxn}$) eine strikt diagonaldominante Matrix. Dann ist $A$ invertierbar und es gilt:*

$$\|Az\|_\infty \geq c\|z\|_\infty \quad \textit{für alle} \quad z \in \mathbb{R}^n \quad (\textit{oder } z \in \mathbb{C}^n)$$

*mit*

$$c := \min_{1\leq i\leq n} \left( |a_{ii}| - \sum_{j\neq i}^{n} |a_{ij}| \right).$$

*Beweis.* Zum Beweis benutzen wir die Aussage $A$ ist invertierbar genau dann, wenn $Ax = 0$ nur die triviale Lösung besitzt. Es seien $x \in \mathbb{R}^n \setminus \{0\}$ und $b = Ax$. Weiter sei

$k \in \{1, \dots, n\}$ so gewählt, dass $\|x\|_\infty = |x_k|$. Dann gilt

$$\begin{aligned}
\|b\|_\infty = \|Ax\|_\infty &= \max_{1\le i\le n} \left| \sum_{j=1}^{n} a_{ij} x_j \right| \\
&\ge \sum_{j=1}^{n} a_{kj} x_j = \left| a_{kk} \frac{x_k}{|x_k|} + \sum_{j\ne k}^{n} a_{kj} \frac{x_j}{|x_k|} \right| \|x\|_\infty \\
&\ge \left( |a_{kk}| - \sum_{j\ne k}^{n} |a_{kj}| \right) \|x\|_\infty \\
&\ge \min_{1\le i\le n} \left( |a_{ii}| - \sum_{j\ne i}^{n} |a_{ij}| \right) \|x\|_\infty = c\|x\|_\infty > 0.
\end{aligned}$$

Also folgt aus $x \neq 0$, dass $Ax = b \neq 0$ und damit ist $A$ invertierbar. Außerdem gilt $\|Ax\|_\infty \ge c\|x\|_\infty$ für alle $x \in \mathbb{R}^n$. Der Beweis funktioniert für $x \in \mathbb{C}^n$ analog. □

*Bemerkung.* Mit Lemma 2.21 folgt die Lösbarkeit für die Spline-Interpolation mit natürlichen, vollständigen und periodischen Randbedingungen.

**Lemma 2.22.** *Das im 4. Fall auftretende („not-a-knot") lineare Gleichungssystem ist eindeutig lösbar.*

*Beweis.* Wir wollen zeigen, dass $Ax = 0$ nur die triviale Lösung besitzt. Dazu formen wir die Matrix $A$ aus dem 4. Fall zunächst so um, dass wir die Matrix

$$\tilde{A} = \begin{pmatrix}
1 & 1 & 1 & & & & \\
0 & \mu_1 - 2 & \mu_1 - \lambda_1 & & & & \\
 & \mu_2 & 2 & \lambda_2 & & & \\
 & & \ddots & \ddots & \ddots & & \\
 & & & \mu_{n-2} & 2 & \lambda_{n-2} & \\
 & & & & \lambda_{n-1} - \mu_{n-1} & \lambda_{n-1} - 2 & 0 \\
 & & & & 1 & 1 & 1
\end{pmatrix}$$

erhalten. Bezeichne $A_i$ die $i$-Zeile von $A$. Dann gilt nämlich $\tilde{A}_1 = A_1 + A_2$ und unter Verwendung von $\mu_1 + \lambda_1 = 1$ gilt $\tilde{A}_2 = \mu_1 A_1 - \lambda_1 A_2$. Analog erhält man die letzten beiden Zeilen von $\tilde{A}$. Wir betrachten nun $\tilde{A}x = 0$. Dann folgt aus

$$\left(\begin{array}{c|ccccc|c}
1 & 1 & 1 & 0 & \cdots & 0 & 0 \\ \hline
0 & & & & & & 0 \\
\vdots & & & A_{22} & & & \vdots \\
0 & & & & & & 0 \\ \hline
0 & 0 & \cdots & 0 & 1 & 1 & 1
\end{array}\right)
\begin{pmatrix} x_1 \\ \vec{x_2} \\ x_3 \end{pmatrix} = 0$$

mit Lemma 2.21 zunächst $\vec{x}_2 = 0$, da $A_{22}$ strikt diagonaldominant ist.. Weiterhin sehen wir dann aber anhand der ersten und letzten Zeile, dass gerade $x_1+0 = 0$ und $x_3+0 = 0$ ist. Also hat $Ax = 0$ nur die triviale Lösung und $A$ ist invertierbar. □

Im folgenden Satz fassen wir nochmal alle betrachteten Interpolationsprobleme zusammen.

**Satz 2.23.** *Die Interpolationsprobleme mit kubischen*

*1) natürlichen Splines*

$$\begin{aligned} &s(x_j) = f(x_j), \quad j = 0, \dots, n, \\ &M_0 = M_n = 0 \quad (\textit{oder } M_0, M_n \textit{ beliebig}), \end{aligned}$$

*2) vollständigen Splines*

$$\begin{aligned} &s(x_j) = f(x_j), \quad j = 0, \dots, n, \\ &s'(x_0) = f'(x_0), \\ &s'(x_n) = f'(x_n), \end{aligned}$$

*3) periodischen Splines*

$$\begin{aligned} &f(x_0) = f(x_n), \\ &s(x_j) = f(x_j), \quad j = 0, \dots, n, \\ &s'(x_0) = s'(x_n), \\ &s''(x_0) = s''(x_n), \end{aligned}$$

*4) not-a-knot-Splines*

$$\begin{aligned} &s(x_j) = f(x_j), \quad j = 0, \dots, n, \\ &s'''(x_1 - 0) = s'''(x_1 + 0), \\ &s'''(x_{n-1} - 0) = s'''(x_{n-1} + 0) \end{aligned}$$

*sind stets eindeutig lösbar.*

*Beweis.* Die entsprechenden lineare Gleichungssysteme sind nach Lemma 2.21 und Lemma 2.22 eindeutig lösbar. Folglich existiert $s''$ eindeutig. Also existiert ein $s$, dass die Interpolationsaufgabe löst. Ist $\tilde{s} \in S_3(X)$ eine weitere Lösung, so folgt $s - \tilde{s} = cx + d$, da $s'' = \tilde{s}''$. Aus der Interpolationsbedingung für $j = 0$ und $j = n$ folgt dann aber $s = \tilde{s}$. □

### 2.3.2 Konvergenz kubischer Splines

Es seien $X : a = x_0 < \cdots < x_n = b$ eine Zerlegung von $I = [a, b]$ sowie

$$h := \max_{j=1,\ldots,n} (x_j - x_{j-1}) \qquad \text{der maximale Gitterabstand und}$$

$$h_{min} := \min_{j=1,\ldots,n} (x_j - x_{j-1}) \qquad \text{der minimale Gitterabstand.}$$

Wir sind nun am Verhalten des Fehlers $f - s$ für $n \to \infty$ interessiert.

**Satz 2.24.** *Es sei $f \in C^4([a, b])$ mit $f''(a) = f''(b) = 0$. Es sei weiter $s$ der kubische natürliche Spline, der $f$ auf der Zerlegung $X$ interpoliert. Dann gilt die Fehlerabschätzung*

$$\|f - s\|_{\infty,[a,b]} \leq h^4 \|f^{(4)}\|_{\infty,[a,b]}.$$