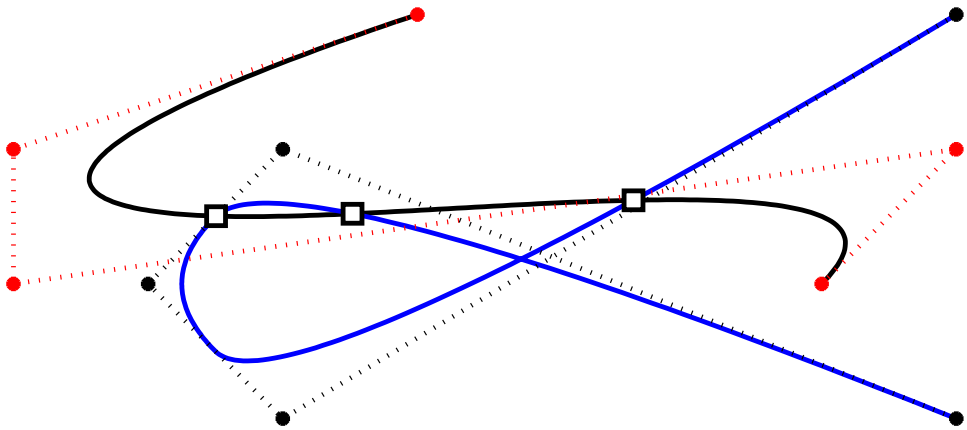
Stefan Funken, Dirk Lebiedz, Karsten Urban

Numerik II

(Einführung in die Numerische Analysis)



SKRIPT, UNIVERSITÄT ULM, SOMMERSEMESTER 2013

Vorwort. Dieses Manuskript ist entstanden aus Mitschriften und Skripten verschiedener Vorlesungen, die wir seit 2002 an der Universität Ulm gehalten haben. Es ist der Sinn des vorliegenden Dokumentes, den Studierenden unserer Vorlesungen einen einheitlichen Stoffumfang für die Vorlesung *Numerik II* zu geben, unabhänging davon, wer von uns tatsächlich die Vorlesung hält. In diesem Sinne bietet das vorliegende Manuskript einen Rahmen für die Nachbearbeitung der Vorlesungen und der Vorbereitung auf Prüfungen. Dieses Manuskript kann keinesfalls das Studium von Lehrbüchern ersetzen. Eine entsprechende Liste von Lehrbüchern findet sich im Literaturverzeichnis und auf der Internet–Seite der Vorlesung.
Jedes Manuskript weist Fehler auf, sicher auch dieses. Wenn Sie Fehler, Druckfehler, sprachliche Unzulänglichkeiten oder inhaltliche Flüchtigkeiten finden, würden wir uns über einen entsprechenden Hinweis per Email freuen. Sie helfen damit zukünftigen Studierenden. Vielen Dank im Voraus.

Danksagung. Einige Vorgängerversionen dieses Manuskriptes wurden aus Mitteln der Studiengebühren finanziert. Eine Reihe von Personen haben bei der Erstellung geholfen. Wir danken Theresa und Julia Springer, Markus Bantle und Judith Rommel für zahlreiche Hinweise. Frau Kristin Kirchner und Herrn Moritz Reinhard sind wir für das sorgfältige Lesen des Manuskripts zu besonderem Dank verpflichtet. Ihre zahlreichen Kommentare, Vorschläge, Korrekturen und Hinweise haben die Qualität des Textes wesentlich verbessert.

Ganz besonderer Dank gebührt auch Frau Petra Hildebrand, die unsere handschriftlichen Aufzeichnungen in LaTeX umgesetzt und zahlreiche Grafiken erstellt hat.

Die hier vorliegende Version des Manuskripts wurde von Frau Katharina Becker-Steinberger grundlegend überarbeitet, vereinheitlicht und inhaltlich ergänzt, dafür danken wir ganz herzlich. Ihre umsichtigen Anmerkungen und Korrekturen haben wesentlich dazu beigetragen das aktuell Manuskript zu erstellen.

Stand. Ulm, Juni 2013.
Stefan Funken, Dirk Lebiedz, Karsten Urban

Inhaltsverzeichnis

1 NICHTLINEARE GLEICHUNGEN

Nachdem wir uns in der Vorlesung Numerik I mit dem Lösen linearer Gleichungssysteme beschäftigt haben, wenden wir uns nun nichtlinearen Gleichungen (in einer oder mehreren Variablen) zu. Diese Gleichungen treten häufig als Teilaufgabe bei der Behandlung komplexerer Probleme auf.

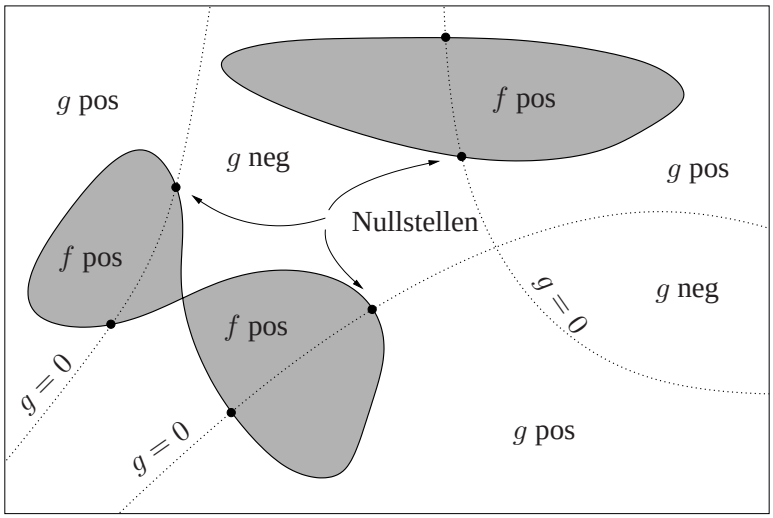


Abb. 1.1: Lösung von zwei Gleichungen in zwei Unbekannten. Die durchgezogenen Linien sind Niveaulinien zu $f(x, y) = 0$, die gestrichelten Linien zu $g(x, y) = 0$. Die gesuchten Lösungen sind die Schnittpunkte der völlig unabhängigen Nulllinien. Die Anzahl der Nullstellen ist im Allgemeinen a-priori nicht bekannt.

1.1 PROBLEMSTELLUNG UND BEISPIELE

In der Numerik 1 haben wir lineare Gleichungssysteme

$$Ax = b, \quad A \in \mathbb{R}^{n\times n}, b \in \mathbb{R}^n$$

betrachtet, d.h. in diesem Fall hatten wir ein lineare Funktion $A : \mathbb{R}^n \to R^n$. Ist nun allgemein ein System von n nichtlinearen Gleichungen in n Unbekannten gegeben, d.h. mit einer stetigen, nichtlinearen Funktion

$$\tilde{F} : \mathbb{R}^n \to \mathbb{R}^n,$$

so können wir das gegebene Problem $\tilde{F}(x) = b$ in eine **Nullstellenaufgabe** transformieren, so dass Lösungen $x \in \mathbb{R}^n$ der Gleichung

$$F(x) := \tilde{F}(x) - b = 0 \tag{1.1}$$

zu bestimmen sind. Noch allgemeiner: $F : \Omega \to W$ mit Mengen $\Omega, W \subset \mathbb{R}^n$. Wir beschränken uns aber in dieser Vorlesung auf den Fall $W = \mathbb{R}^n$. Für

- $n = 1$ spricht man von einer *skalaren* Gleichung,
- für $n > 1$ von einem *System*.

Zur Motivation stellen wir zunächst mehrere Modellbeispiele, bei denen nichtlineare Gleichungen auftreten, vor.

Beispiel 1.1.1 (Erste einfache Beispiele) Wir beginnen mit einigen einfachen Beispielen, die aber bereits einen Eindruck von der Vielfalt nichtlinearer Gleichungssysteme vermitteln.

(a) Löse $f(x) = 0$ für $f(x) = x^2 - 2px + q$ mit $p, q \in \mathbb{R}$. Bekanntermaßen gibt es zwei Lösungen $x_{1,2} = p \pm \sqrt{p^2 - q}$. Falls $p^2 - q < 0$ gibt es keine reelle Lösung. An diesem ganz einfachen Beispiel sieht man bereits, dass Existenz und Eindeutigkeit im nichtlinearen Fall nicht trivial sind.

(b) Es müssen aber nicht immer Zahlen (oder Vektoren) die gesuchten Lösungen sein. Als Beispiel betrachte man nichtlineare Differentialgleichungen, z.B. die Transportgleichung („Burgers–Gleichung“):

$$\frac{d}{dt}u(t,x) + u(t,x)\frac{d}{dx}u(t,x) = 0, \quad \text{(kurz:} \quad u_t + uu_x = 0)$$

wobei $u(t,x)$ die Geschwindigkeit eines Teilchens zur Zeit t am Ort x ist. Die Unbekannte ist also eine *Funktion*, man sucht in einem *unendlich-dimensionalen* Raum (den Raum aller in t und x stetig differenzierbareb Funktionen). Anwendungen sind z.B. Simulationen des Straßenverkehrs oder Informationsausbreitung im Internet.

(c) Numerische Transformation von Wahrscheinlichkeits–Verteilungsfunktionen und die Berechnung von Quantilen. Aus der Wahrscheinlichkeitsrechnung kennen Sie Tabellen von Quantilen z.B. der Standardnormalverteilung. Diese können nicht exakt berechnet werden, sondern ergeben sich als Lösung eines nichtlinearen Problems.

(d) Fast alle „realen“ Phänomene sind nichtlinear — werden oft zur Vereinfachung linearisiert.

Wie man schon an obigem Beispiel sieht, hat man hier keine „handlichen“ Kriterien für Existenz und Eindeutigkeit. Dies hat natürlich auch Konsequenzen für die numerischen Lösungsverfahren. Im Folgenden zeigen wir weitere Beispiele.

Beispiel 1.1.2 (Zinssatz bei einem Kredit) Wie hoch darf der Zinssatz sein, wenn man einen Kredit über 10.000 Euro in 10 Jahren abzahlen möchte und man höchstens 250 Euro monatlich aufbringen kann? Oder allgemeiner eine Kreditsumme K_0 in n Jahren mit monatlichen Raten R getilgt haben möchte?
Der Einfachheit halber rechnen wir den jährlichen Zinssatz p in einen monatlichen Zinssatz m um, d.h. verzinst man monatlich mit einem Prozentsatz m oder jährlich mit p, so erhält man am Ende des Jahres jeweils das Gleiche. Die Beziehung zwischen p und m ist also $(1+m)^{12} = 1+p$.
Für den Restbetrag K_i des Kredits nach $i \in \mathbb{N}$ Monaten gilt:

$$\begin{aligned}
K_i &= K_{i-1}(1+m) - R, \\
K_{i+1} &= K_i(1+m) - R = \left[K_{i-1}(1+m) - R\right](1+m) - R, \\
&= K_{i-1}(1+m)^2 - R\left[(1+m) + 1\right], \\
&\vdots \\
K_n &= K_0(1+m)^n - R\left[(1+m)^{n-1} + \ldots + (1+m) + 1\right], \\
&= K_0(1+m)^n - R\left[(1+m)^n - 1\right]/m, \\
&= (K_0 - R/m)(1+m)^n + R/m.
\end{aligned}$$

Zu gegebener Kreditsumme K_0, monatlichem Zinssatz m und Tilgungsdauer (n Monate) berechnet sich die Rate R, um den Kredit vollständig zu tilgen, indem wir $K_n = 0$ setzen, d.h.

$$R = K_0\, m\, (1+m)^n / \left[(1+m)^n - 1\right] .$$

Auch die Tilgungsdauer lässt sich analytisch darstellen

$$n = \frac{\log\left(R/(R - m\,K_0)\right)}{\log(1+m)}\,.$$

(Wie ist hier das Ergebnis zu interpretieren, falls n nicht ganzzahlig ist?) Wie gewinnt man jedoch m zu gegebenen $K_0 > 0$, $n \in \mathbb{N}$, $K_0 > R > K_0/n$ aus der Gleichung

$$(m\,K_0 - R)(1+m)^n + R = 0\,?$$

Es ist offensichtlich, dass $0 < m < 1$ gilt und $f(m) := (m\,K_0 - R)(1+m)^n + R$ nur eine Nullstelle in $(0,1)$ hat. Aber wie kann man diese einfach, schnell und numerisch stabil bestimmen?

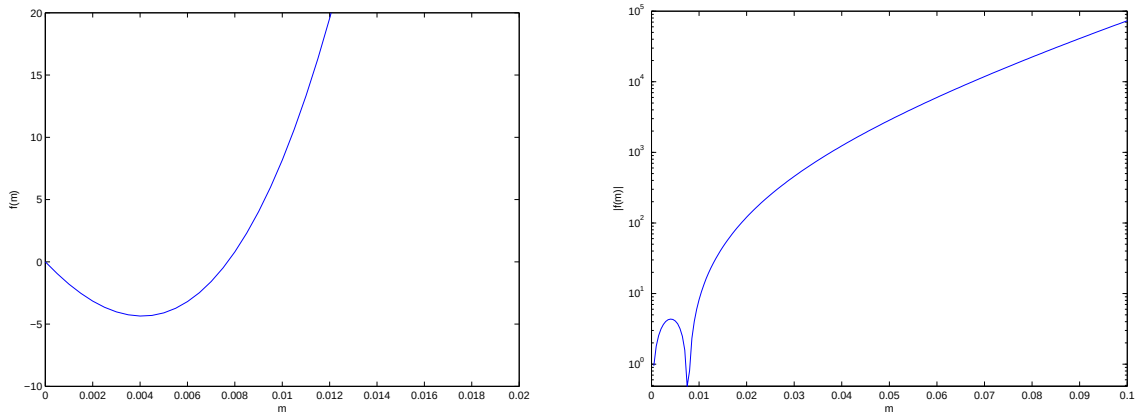


Abb. 1.2: Der Graph von $f(m) := (m\,K_0 - R)(1+m)^n + R$ (links, y-Achse linear skaliert, $x \in [0, 0.02]$) bzw. $|f(m)|$ (rechts, logarithmisch-skaliert, $x \in [0, 0.1]$) für $K_0 = 10000$, $R = 250$ und $n = 48$. Die Funktion f hat eine Nullstelle bei 0 und bei ≈ 0.008.

Beispiel 1.1.3 (Nullstellen von Orthogonalpolynomen) Sei $\mathbb{P}_n$ der Vektorraum der Polynome vom Grad kleiner gleich n. Die ***Legendre*-Polynome** $P_n \in \mathbb{P}_n$ $(n = 0, 1, 2, \ldots)$ erfüllen die Drei-Term-Rekursion

$$P_0(x) = 1,\ P_1(x) = x,\ (n+1)P_{n+1}(x) = (2n+1)xP_n(x) - nP_{n-1}(x),\ \ n \in \mathbb{N}\,.$$

Die ersten Legendre-Polynome lauten

$$\begin{aligned} P_0 &= 1, & P_3 &= \frac{1}{2}(5x^3 - 3x),\\ P_1 &= x, & P_4 &= \frac{1}{8}(35x^4 - 30x^2 + 3),\\ P_2 &= \frac{1}{2}(3x^2 - 1), & P_5 &= \frac{1}{8}(63x^5 - 70x^3 + 15x)\,. \end{aligned}$$

Die Nullstellen der *Legendre*-Polynome liegen in $(-1, 1)$ und die Nullstellen von P_n trennen die Nullstellen von P_{n+1} $(n \in \mathbb{N})$. Mit Hilfe dieser Eigenschaft lassen sich sukzessive Intervalle finden, in denen Nullstellen liegen, z.B. hat P_1 die Nullstelle $\xi_1^{(1)} = 0$ und somit liegen die Nullstellen $\xi_1^{(2)}$, $\xi_2^{(2)}$ von P_2 in $(-1, 0)$ und $(0, 1)$, die Nullstellen $\xi_1^{(3)}$, $\xi_2^{(3)}$, $\xi_3^{(3)}$ von P_3 in $(-1, \xi_1^{(2)})$, $(\xi_1^{(2)}, \xi_2^{(2)})$ und $(\xi_2^{(2)}, 1)$. Diese Eigenschaft der *Legendre*-Polynome gilt auch für weitere Orthogonalpolynome (siehe Satz **??** in Kapitel 3). Die Berechnung der Nullstellen ist u.a. wichtig im Zusammenhang mit *Gauß*-Quadraturformeln.

Beispiel 1.1.4 (Extremalstellen von skalaren Funktionen) Die mehrdimensionale Erweiterung der *Rosenbrock*[1]-Funktion $f : \mathbb{R}^n \to \mathbb{R}$ mit

$$f(x) = \sum_{i=1}^{n-1} \left[(1 - x_i)^2 + 100(x_{i+1} - x_i^2)^2\right], \quad x \in \mathbb{R}^n$$

hat für $n \geq 4$ mindestens ein lokales Minimum in der Umgebung von $(x_1, x_2, \ldots, x_n)^T = (-1, 1, \ldots, 1)^T$ neben dem globalen Minimum $(x_1, \ldots, x_n)^T = (1, \ldots, 1)^T$. Diese Extremalstellen erfüllen notwendigerweise die Gleichung $\nabla f = 0$. Wie kann man nun für $n \geq 4$ ein solches lokales Minimum bestimmen? Dies bedeutet, ein $x \in \mathbb{R}^n \setminus \{(1, \ldots, 1)^T\}$ ist zu bestimmen mit

$$\begin{aligned}
g(x) &= (g_1(x), \ldots, g_n(x))^T = 0, \\
g_1(x) &:= \frac{\partial f}{\partial x_1} = -2(1 - x_1) - 400x_1(x_2 - x_1^2), \\
g_i(x) &:= \frac{\partial f}{\partial x_i} = -2(1 - x_i) - 400x_i(x_{i+1} - x_i^2) + 200(x_i - x_{i-1}^2), \quad i = 2, \ldots, n-1\,, \\
g_n(x) &:= \frac{\partial f}{\partial x_n} = 200(x_n - x_{n-1}^2)\,.
\end{aligned}$$

1.1.1 Notation

Sei X der zugrunde liegende *Banach*-Raum. Außerdem bezeichnen wir mit $\|\cdot\|$ eine beliebige Norm auf X. Wir betrachten hier die Fälle

- $X = \mathbb{R}$ versehen mit der Betrags-Norm $\|\cdot\| \equiv |\cdot|$ und
- $X = \mathbb{R}^n$.

Vektoren $x \in \mathbb{R}^n$ werden hier grundsätzlich als **Spaltenvektoren** verstanden. Mit x^T bezeichnen wir den durch Transposition entstehenden Zeilenvektor.

Wenn nicht explizit anders erwähnt, verwenden wir auf $X = R^n$ die ***Euklidische* Norm** $\|\cdot\| \equiv \|\cdot\|_2$ mit:

$$\|x\|_2 = \sqrt{x^T x} = \left(\sum_{i=1}^{n} x_i^2\right)^{1/2}, \quad x \in \mathbb{R}^n.$$

Weitere nützliche p-Normen sind hier die Betragssummennorm

$$\|x\|_1 := \sum_{i=1}^{n} |x_i|$$

und die Maxiumsnorm

$$\|x\|_\infty = \max_{1 \leq i \leq n} |x_i|.$$

Sei $\epsilon > 0$ und $x^* \in \mathbb{R}^n$, dann bezeichnen wir die **offene ϵ-Umgebung um** x^* mit $K_\epsilon(x^*)$:

$$K_\epsilon(x^*) := \{x \in \mathbb{R}^n : \|x^* - x\| < \epsilon\}.$$

Ist die skalarwertige Funktion $f : \mathbb{R}^n \to \mathbb{R}$ stetig differenzierbar, dann wird

$$\nabla f(x) = \begin{pmatrix} \frac{\partial f}{\partial x_1}(x) \\ \vdots \\ \frac{\partial f}{\partial x_n}(x) \end{pmatrix} \in \mathbb{R}^n$$

[1] Rosenbrock, Howard Harry (1920 - 2010)

der **Gradient** von f im Punkt x genannt. Ist f zweimal stetig differenzierbar, dann nennt man

$$\nabla^2 f(x) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1 \partial x_1}(x) & \dots & \frac{\partial^2 f}{\partial x_1 \partial x_n}(x) \\ \vdots & & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1}(x) & \dots & \frac{\partial^2 f}{\partial x_n \partial x_n}(x) \end{pmatrix} \in \mathbb{R}^{n \times n}$$

die **Hesse-Matrix** von f im Punkt x.

Ist $F : \mathbb{R}^n \to \mathbb{R}^n$ stetig differenzierbar, dann ist

$$J_F(x) = \begin{pmatrix} \nabla F_1(x)^T \\ \vdots \\ \nabla F_n(x)^T \end{pmatrix} = \begin{pmatrix} \frac{\partial F_1}{\partial x_1}(x) & \dots & \frac{\partial F_1}{\partial x_n}(x) \\ \vdots & & \vdots \\ \frac{\partial F_n}{\partial x_1}(x) & \dots & \frac{\partial F_n}{\partial x_n}(x) \end{pmatrix} \in \mathbb{R}^{n \times n}$$

die ***Jacobi*-Matrix** von F im Punkt x.

1.1.2 Grundlegende Begriffe: Konvergenzgeschwindigkeit

Wir betrachten stets eine Folge als ein iteratives Verfahren zur Approximation einer Unbekannten x^*, z.B. der Lösung von $F(x) = 0$. Also möchten wir natürlich, dass

$$x^{(k)} \to x^*, \quad k \to \infty$$

möglichst schnell konvergiert. Oder, anders ausgedrückt, bei vorgegebenen $\epsilon > 0$ (einer Toleranz) möchten wir möglichst wenige Schritte unternehmen, also

$$\|x^{(k)} - x^*\| \leq \epsilon$$

mit einem möglichst kleinen $n = n(\epsilon) \in \mathbb{N}$. Dies führt auf den Begriff der *Konvergenzordnung*, die ein Maß für die Konvergenzgeschwindigkeit ist.

Definition 1.1.5 (Konvergenzgeschwindigkeit) *Sei $(x^{(k)})_{k\in\mathbb{N}}$ eine konvergente Folge mit $x^{(k)} \in \mathbb{R}^n$ und Grenzwert $x^* \in \mathbb{R}^n$.*

1. *Man bezeichnet $(x^{(k)})_{k\in\mathbb{N}}$ als (mindestens)* **linear konvergent** *(oder Q-**linear konvergent**), wenn es ein $0 < \rho < 1$ gibt mit*

   $$\|x^{(k+1)} - x^*\| \leq \rho \|x^{(k)} - x^*\|, \quad k = 0, 1, \dots .$$

   Die Zahl ρ wird **Konvergenzfaktor** *(oder auch* **Kontraktionsrate***) genannt.*

2. *Gibt es eine gegen Null konvergente Folge $(\rho_k)_{k\in\mathbb{N}}$ mit*

   $$\|x^{(k+1)} - x^*\| \leq \rho_k \|x^{(k)} - x^*\|, \quad k = 0, 1, \dots ,$$

   so heißt $(x^{(k)})_{k\in\mathbb{N}}$ (mindestens) **superlinear konvergent** *(oder Q-**superlinear konvergent**).*

3. *Gibt es ein $\rho > 0$ und ein $1 < q \in \mathbb{R}$ mit*

   $$\|x^{(k+1)} - x^*\| \leq \rho \|x^{(k)} - x^*\|^q, \quad k = 0, 1, \dots ,$$

   so heißt $(x_k)_{k\in\mathbb{N}}$ konvergent mit (mindestens) **Konvergenzordnung** *q. Für $q = 2$ spricht man auch von (mindestens)* **quadratischer Konvergenz** *(oder Q-**quadratischer Konvergenz**).*

4. *Man nennt $(x_k)_{k\in\mathbb{N}}$ R-***linear konvergent** *mit Rate $0<\gamma<1$ gegen x^*, falls es eine Folge $(\alpha_k)_{k\in\mathbb{N}}\subset(0,\infty)$ gibt, die Q-linear mit Rate γ gegen 0 konvergiert, so dass gilt:*

$$\|x^{(k)}-x^*\|\le\alpha_k \quad \text{für } k\to\infty.$$

5. *Die Folge $(x_k)_{k\in\mathbb{N}}$ heißt R-***superlinear konvergent** *mit Rate $0<\gamma<1$ gegen x^*, falls es eine Folge $(\alpha_k)_{k\in\mathbb{N}}\subset(0,\infty)$ gibt, die Q-superlinear mit Rate γ gegen 0 konvergiert, so dass gilt:*

$$\|x^{(k)}-x^*\|\le\alpha_k \quad \text{für } k\to\infty.$$

6. *Die Folge $(x_k)_{k\in\mathbb{N}}$ heißt R-***quadratisch konvergent** *gegen x^*, falls es eine Nullfolge $(\alpha_k)_{k\in\mathbb{N}}\subset(0,\infty)$ gibt, die Q-quadratisch gegen 0 konvergiert, so dass gilt:*

$$\|x^{(k)}-x^*\|\le\alpha_k \quad \text{für } k\to\infty.$$

Bemerkung 1.1.6 (Charakterisierung der Q-Konvergenz via *Landau*'scher-Symbolik) *Mit der o- und $\mathcal{O}$-Notation kann man die Definitionen auch wie folgt schreiben: Eine Folge $(x^{(k)})_{k\in\mathbb{N}}$ mit $x^{(k)}\to x^*$ heißt (mindestens) Q-***superlinear konvergent***, falls*

$$\|x^{(k+1)}-x^*\|=o(\|x^{(k)}-x^*\|), \quad \text{für } k\to\infty,$$

*die Folge heißt (mindestens) Q-***quadratisch konvergent***, falls*

$$\|x^{(k+1)}-x^*\|=\mathcal{O}(\|x^{(k)}-x^*\|^2), \quad \text{für } k\to\infty.$$

Bemerkung 1.1.7 (Charakterisierung der Q-Konvergenz über Brüche) *Eine Folge $(x^{(k)})_{k\in\mathbb{N}}$ mit $x^{(k)}\to x^*$ wird auch als Q-***superlinear konvergent** *bezeichnet, falls*

$$\lim_{k\to\infty}\frac{\|x^{(k+1)}-x^*\|}{\|x^{(k)}-x^*\|}=0$$

*gilt und als Q-***quadratisch konvergent***, falls*

$$\limsup_{k\to\infty}\frac{\|x^{(k+1)}-x^*\|}{\|x^{(k)}-x^*\|}<\infty.$$

Diese Charakterisierungen sind allerdings nur möglich, wenn $x^{(k)}\neq x^*$ für alle $k\in\mathbb{N}$ gilt.

Motiviert durch diese Beschreibung definiert man

Definition 1.1.8 (Q-und R-Faktor) *1. Q-Konvergenz:*

(a) Für eine Folge $(x^{(k)})_{k\in\mathbb{N}}$ mit $x^{(k)}\to x^*$ und $p\in[1,\infty)$ wird

$$Q_q(x^{(k)}):=\begin{cases}\limsup_{k\to\infty}\frac{\|x^{(k+1)}-x^*\|}{\|x^{(k)}-x^*\|^q}, & \text{falls } x^{(k)}\neq x^* \quad \text{für alle } k\in\mathbb{N},\\ 0, & \text{falls } x^{(k)}=x^* \quad \text{für alle } k\ge k_0,\\ +\infty, & \text{sonst.}\end{cases}$$

der **Quotienten-Konvergenzfaktor (Q-Faktor)** *von $(x^{(k)})_{k\in\mathbb{N}}$ genannt.*

(b) Die Größe

$$O_Q(x^{(k)}):=\inf\{p\in[1,+\infty)\ \ Q_p(x^{(k)})=\infty\}$$

*heißt Q-***Konvergenzordnung** *der Folge $(x^{(k)})_{k\in\mathbb{N}}$.*

2. *R-**Konvergenz:***

   (a) *Für eine Folge $(x^{(k)})_{k\in\mathbb{N}}$ mit $x^{(k)} \to x^*$ und $p \in [1,\infty)$ wird*

$$R_p(x^{(k)}) := \begin{cases} \limsup_{k\to\infty} \|x^{(k)} - x^*\|^{1/k}, & \text{falls } p = 1, \\ \limsup_{k\to\infty} \|x^{(k)} - x^*\|^{1/p^k}, & \text{falls } p > 1, \end{cases}$$

   der **Wurzel-Konvergenzfaktor (R-Faktor, engl. root-factor)** *von $(x^{(k)})_{k\in\mathbb{N}}$ genannt.*

   (b) *Die Größe*

$$R_p(x^{(k)}) := \inf\{p \in [1,+\infty) \;\; R_p(x^{(k)}) = 1\}$$

   *heißt R-**Konvergenzordnung** der Folge $(x^{(k)})_{k\in\mathbb{N}}$.*

Bemerkung 1.1.9 *Die Definition Q-Konvergenz basiert somit auf dem Quotientenkriterium für Reihen, während die R-Konvergenz auf dem Wurzelkriterium für Reihen basiert. Offenbar impliziert Q-Konvergenz die entsprechende R-Konvergenz.*

Bemerkung 1.1.10 *Für $p = 1$ und $p = 2$ gilt*

$Q_1(x^{(k)}) = 0$	*Q-superlineare Konvergenz*
$0 < Q_1(x^{(k)}) < 1$	*Q-lineare Konvergenz*
$0 < Q_2(x^{(k)}) < +\infty$	*Q-quadratische Konvergenz*

1.2 Skalare Nichtlineare Gleichungen

Betrachten wir zunächst die Situation in einer Raumdimension. Mit anderen Worten, in diesem Abschnitt beschäftigen wir uns zunächst mit Verfahren zur Lösung einer skalaren nichlinearer Gleichung

$$f(x) = 0.$$

wobei $I := [a,b] \subset \mathbb{R}$ und $f : I \to \mathbb{R}$ eine stetige Funktion.

1.2.1 Bisektionsmethode

Die Bisektionsmethode ist ein sehr einfaches robustes Verfahren, das jedoch nur für *skalare* Gleichungen geeignet ist.

Herleitung des Algorithmus: Diese Methode, motiviert durch Überlegungen aus der reellen eindimensionalen Analysis (siehe Intervallschachtelung), löst (1.1) dadurch, dass sie die Lösung durch systematisch kleiner werdende Intervalle einschließt. Man geht von der Annahme aus, es sei ein Intervall $I := [a,b]$ bekannt mit $f(a) \cdot f(b) < 0$. Da wir f stetig vorrausgesetzt haben können wir den Zwischenwertsatz anwenden. Mit anderen Worten, die Annahme $f(a) \cdot f(b) < 0$ liefert die Existenz eine Nullstelle $x^* \in (a,b)$, d.h. die Existenz eines x^* im Inneren von I mit $f(x^*) = 0$.

Zunächst wird in den Mittelpunkt $m = \frac{1}{2}(a+b)$ des Intervalls I der Funktionswert $f(m)$ bestimmt. Ist $f(m) \neq 0$ entscheidet nun das Vorzeichen, in welchem der Teilintervalle $[a,m], [m,b]$ die gesuchte Lösung x^* liegt. Nun passt man die Intervallgrenzen entsprechend an, so dass x^* im neuen halbierten Intervall liegt und beginnt von vorne.

Etwas mathematischer: Wir bezeichnen die Grenzen des Ausgangsintervalls I mit $a_0 := a$ und $b_0 := b$ wählen als Startwert den Mittelwert $x^{(0)} := \frac{1}{2}(a_0 + b_0)$. Führen einen Vorzeichentest durch, um zu prüfen, auf welcher Seite des Mittelpunkts die Nullstelle liegt. Falls $f(x^{(0)})f(a_0) < 0$ ist wählen wir das linke Teilintervall als neues Intervall und setzen $a_1 := a_0, b_1 := x^{(0)}$, ansonsten, wählen wir das rechte Teilintervall und setzen $a_1 := x^{(0)}, b_1 := b_0$. Von dem neuen Teilintervall bestimmen wir unsere nächste Iterierte wiederum als Mittelpunkt $x^{(1)} = \frac{1}{2}(a_1 + b_1)$. Wir erhalten damit folgenden Algorithmus:

Algorithmus 1.2.1 (Bisektionsmethode) *Seien $I := [a, b]$ mit $f(a)f(b) < 0$, $\epsilon > 0$ gegeben und setze $a_0 := a, b_0 := b$.*
Für $k = 0, 1, 2 \ldots$:

1) *Berechne $x^{(k)} = \frac{1}{2}(a_k + b_k)$ und $f(x^{(k)})$.*

2) *Falls $|f(x^{(k)})| \leq \epsilon$, STOPP.*

3) *Falls $f(x^{(k)})f(a_k) < 0$, setze*

$$a_{k+1} := a_k, \quad b_{k+1} := x^{(k)},$$

sonst

$$a_{k+1} := x^{(k)}, \quad b_{k+1} := b_k.$$

Bemerkung 1.2.1 (Abbruchbedingungen) *Sei $\epsilon > 0$ eine gegebene Toleranz. Man könnte an die folgenden zwei Abbruchbedingungen für das Bisektionsverfahren denken: Einerseits das in Algorithmus 1.2.1 Schritt 2 verwendete Abbruchkriterium, bei dem gefordert wird, dass der Funktionswert in der aktuellen Näherung betragsmäßig sehr klein wird:*

$$|f(x^{(k)})| \leq \epsilon.$$

Und andererseits, das im nachfolgenden Matlab-Programm verwendete Abbruchkriterium, bei dem der Algorithmus abbricht, wenn das Intervall sehr klein wird:

$$|a_k - b_k| \leq \epsilon.$$

MATLAB-Funktion: BisektionsMethode.m

```
function Nullstelle = BisektionsMethode(a,b,func,epsilon)
% Initialisierung
temp=[]
fa = func(a); fb = func(b);
while abs(a-b) > epsilon
 m = (a+b)/2;
 fm = func(m);
 if fm == 0
   Nullstelle = m;
   return
 elseif fa*fm < 0
   b = m;
   fb = fm;
 else
   a = m;
   fa = fm;
```

```
17   end
18   temp = [temp;m];
19  end
20  % Lösung
21  Nullstelle = m;
```

MATLAB-Beispiel:

Testen wir nun die Bisektionsmethode anhand von Bsp. 1.1.2. Wie hoch darf der Zinssatz sein, wenn man einen Kredit über 10.000 Euro in 48 Monaten zurückzahlen möchte, aber nur 250 Euro monatlich aufbringen kann?

```
>> n = 48;
>> K0 = 10000;
>> R = 250;
>> f = @(m) (m*K0-R)*(1+m)^n+R;
>> m = Bisektionsmethode(eps,1,f,1e-7)
m =
   0.00770145654678
>> p = 100 * ((1+m)^12-1)
p =
   9.64343564476941
```

Nach der Umrechnung in den jährlichen Zinssatz sehen wir, dass wir uns den Kredit nur erlauben könnten, wenn der Zinssatz niedriger als 9.65% ist.

Offenbar wird die Intervalllänge in jeder Iteration halbiert. Die Methode konvergiert somit. Gleichzeitig erhalten wir eine a-priori Fehlerabschätzung. Dies halten wir in der folgenden Bemerkung fest.

Bemerkung 1.2.2 (Konvergenz des Bisektionsverfahrens) *Betrachten wir den Mittelpunkt $x^{(k)}$ des Intervalls nach der k-ten Intervallhalbierung als Näherung an x^*, so gilt die a-priori Fehlerabschätzung*

$$|x^{(k)} - x^*| \leq \frac{b-a}{2^k}, \quad k = 0, 1, 2, \ldots.$$

Da die Fehlerschranke wie eine geometrische Folge abnimmt, liest man manchmal, dass die „Konvergenzordnung“ 1 sei (also lineare Konvergenz vorliege). Nach Definition 1.1.5 ist das Bisektionsverfahren nicht *Q-linear konvergent, da auf der rechten Seite der Abschätzung nicht $|x^{(k-1)} - x^*|$ steht, man also keine monotone Reduktion des Fehlers hat. Es gibt Beispiele, bei denen der Fehler springt, vgl. [QSS1, §6.2]. Allerdings ist das Bisektionsverfahren R-linear konvergent, denn die Folge der Beträge der absoluten Fehler $e^{(k)} := x^{(k)} - x^*$, d.h. $(\beta_k)_{k\in\mathbb{N}}$, mit $\beta_k := |e^{(k)}|$, wird durch die linear konvergente Folge $(\alpha_k)_{k\in\mathbb{N}}$, mit $\alpha_k := b - a/2^k$ majorisiert.*

1.2.2 Regula-Falsi[1]

Herleitung des Algorithmus: Wiederum gehen wir davon aus, dass ein Intervall $I := [a, b]$ mit $f(a) \cdot f(b) < 0$ bekannt sei. Anstatt nun I durch Hinzufügen eines Mittelpunkts in zwei Intervalle zu zerlegen, wählen wir eine zusätzliche Stelle ξ, die Nullstelle der Geraden durch $(a, f(a))$ und $(b, f(b))$, d.h.

$$\xi = a - \frac{b-a}{f(b)-f(a)} f(a) = \frac{af(b) - bf(a)}{f(b)-f(a)}. \tag{1.2}$$

Mit den beiden Intervallen $[a, \xi]$, $[\xi, b]$ verfahren wir nun analog zur Methode der Intervallhalbierung. D.h. durch einen Vorzeichentest prüfen wir wieder, in welchem der beiden Teilintervalle

[1] lat.: „Regel des falschen Ansatzes“

die Nullstelle liegt, und nehmen das entsprechende Intervall als neues Intervall. Als Algorithmus

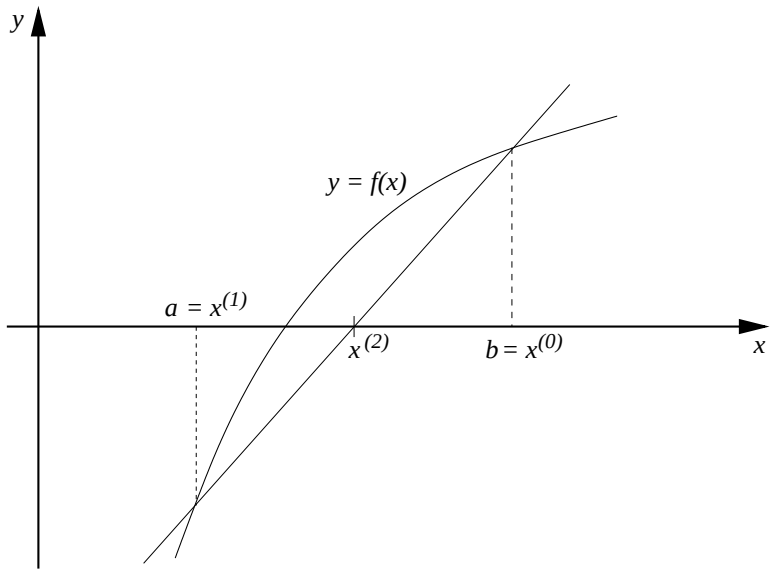


Abb. 1.3: Die geometrische Interpretation der Regula-Falsi-Methode.

ergibt sich somit:

Algorithmus 1.2.2 (Regula-Falsi) *Seien $I := [a, b]$ mit $f(a)f(b) < 0$, $\epsilon > 0$ gegeben und setze $a_0 := a, b_0 := b$.*
Für $k = 0, 1, 2 \ldots$:

1) *Berechne $x^{(k)} = \frac{a_k f(b_k) - b_k f(a_k)}{f(b_k) - f(a_k)}$ und $f(x^{(k)})$.*

2) *Falls $|f(x^{(k)})| \leq \epsilon$, STOPP.*

3) *Falls $f(a_k)f(x^{(k)}) < 0$, setze*

$$a_{k+1} := a_k, \quad b_{k+1} := x^{(k)},$$

sonst

$$a_{k+1} := x^{(k)}, \quad b_{k+1} := b_k.$$

Satz 1.2.3 (Konvergenzordnung Regula-Falsi-Methode) *Es sei x^* einzige Nullstelle von f in $I := [a, b]$, $f \in C^3(I)$ und $f'(x^*) \cdot f''(x^*) \neq 0$. Dann beträgt die Konvergenzordnung der Regula-Falsi-Methode $p = 1$.*

Beweis. Es bezeichne $(x^{(k)})_{k\in\mathbb{N}}$ die sich aus aus obigem Algorithmus 1.2.2 ergebende Folge von „Mittelpunkten“, d.h. $x^{(k)} \in (a_k, b_k)$ ($k \in \mathbb{N}$). Es seien

$$\varepsilon_k^a := a_k - x^*, \quad \varepsilon_k^b := b_k - x^*.$$

Aus (1.2) und der Voraussetzung $f(x^*) = 0$ folgt

$$\begin{aligned} \varepsilon_k := x^{(k)} - x^* &= \frac{(a_k - x^*)f(b_k) - (b_k - x^*)f(a_k)}{f(b_k) - f(a_k)} \\ &= \frac{\varepsilon_k^a f(x^* + \varepsilon_k^b) - \varepsilon_k^b f(x^* + \varepsilon_k^a)}{f(x^* + \varepsilon_k^b) - f(x^* + \varepsilon_k^a)}. \end{aligned}$$

Da $f \in C^3(I)$ nach Voraussetzung gilt, liefert die *Taylor*[2]-Entwicklung:

$$\begin{aligned}
\varepsilon_k &= \frac{\varepsilon_k^a\{\varepsilon_k^b f'(x^*) + \frac{1}{2}(\varepsilon_k^b)^2 f''(x^*) + \ldots\} - \varepsilon_k^b\{\varepsilon_k^a f'(x^*) + \frac{1}{2}(\varepsilon_k^a)^2 f''(x^*) + \ldots\}}{\{\varepsilon_k^b f'(x^*) + \frac{1}{2}(\varepsilon_k^b)^2 f''(x^*) + \ldots\} - \{\varepsilon_k^a f'(x^*) + \frac{1}{2}(\varepsilon_k^a)^2 f''(x^*) + \ldots\}} \\
&= \frac{\frac{1}{2}\varepsilon_k^a\varepsilon_k^b(\varepsilon_k^b - \varepsilon_k^a) f''(x^*) + \ldots}{(\varepsilon_k^b - \varepsilon_k^a)\{f'(x^*) + \frac{1}{2}(\varepsilon_k^b + \varepsilon_k^a) f''(x^*) + \ldots\}} \\
&= \frac{\frac{1}{2}\varepsilon_k^a\varepsilon_k^b f''(x^*) + \ldots}{f'(x^*) + \ldots}
\end{aligned}$$

Nach der Herleitung des Algorithmus gilt $a_k < x^* < b_k$. Demzufolge besitzen ε_k^a und ε_k^b entgegengesetztes Vorzeichen und es gilt weiter $\varepsilon_k^b - \varepsilon_k^a > 0$. Für hinreichend kleine $|\varepsilon_k^a|$ und $|\varepsilon_k^b|$ folgt damit

$$\varepsilon_k \approx \frac{f''(x^*)}{2f'(x^*)}\varepsilon_k^a\varepsilon_k^b\,. \tag{1.3}$$

Mit $f''(x^*) \neq 0$ gilt aber weiterhin, dass f in einer hinreichend kleinen Umgebung von x^* konvex oder konkav ist, und folglich bleibt ein Intervallende fest und wird im Verfahren nur umbenannt. Deshalb ist ε_k nur direkt proportional zu einem der beiden vorhergehenden ε_k^a oder ε_k^b. Die asymptotische Fehlerkonstante C ist für eine konkave Funktion gegeben durch

$$C = \left|\frac{f''(x^*)}{2f'(x^*)}\varepsilon_k^a\right|, \qquad \varepsilon_k^a = \varepsilon_{k-1}^a = \ldots = \varepsilon_1^a = x_1 - x^*,$$

und damit konvergiert die Folge $(x^{(k)})_{k\in\mathbb{N}}$ linear. Analoges gilt für konvexe Funktionen mit ε_k^b anstatt ε_k^a. □

MATLAB-Funktion: RegulaFalsi.m

```
function Nullstelle = RegulaFalsi(a,b,func,epsilon)
fa = func(a); fb = func(b); % Initialisierung
m = (a*fb-b*fa)/(fb-fa); fm = func(m);
while abs(fm) > epsilon
 if fa*fm < 0
   b = m;
   fb = fm;
 else
   a = m;
   fa = fm;
 end
 m = (a*fb-b*fa)/(fb-fa);
 fm = func(m);
end
Nullstelle = m; % Lösung
```

[2] Taylor, Brook (1685-1731)

MATLAB-Beispiel:

Testen wir nun das Regula-Falsi-Verfahren anhand Bsp. 1.1.2 mit $K_0 = 10000$, $n = 48$ und $R = 250$.

```
>> n = 48;
>> K0 = 10000;
>> R = 250;
>> f = @(m) (m*K0-R)*(1+m)^n+R;
>> m = RegulaFalsi(1e-5,0.1,f,1e-7)
m =
   0.00770147244890
```

Bei einem genaueren Vergleich mit dem Bisektionsverfahren stellt man bei diesem Beispiel eine langsamere Konvergenz des Regula-Falsi-Verfahrens fest.

Bemerkung 1.2.4 (Abbruchbedingung für Verfahren 1. Ordnung) *Wir hatten bereits über denkbare Abbruchbedingungen im Rahmen des Bisektionsverfahrens nachgedacht. Allgemeiner formuliert könnte man zu einer gewünschter Zielgenauigkeit $\epsilon > 0$ einerseits an die Bedingung*

$$|x^{(k)} - x^*| \leq \epsilon$$

*und andererseit an die bereits angesprochene Bedingung $|f(x^{(k)})| \leq \epsilon$ denken. Allerdings enthällt erster Abbruchbedingung die unbekannte Nullstelle x^*. Für **Verfahren 1. Ordnung** gilt mit der Dreichecksungleichung:*

$$|x^{(k)} - x^{(k+1)}| \geq |x^{(k)} - x^*| - |x^{(k+1)} - x^*| \geq \left(\frac{1}{C} - 1\right) |x^{(k+1)} - x^*|$$

also

$$|x^{(k+1)} - x*| \leq \frac{C}{1-C}|x^{(k)} - x^{(k+1)}|.$$

Auf der rechten Seite dieser Ungleichung kommt x^* nicht mehr vor! Oftmals verwendet man daher für Verfahren 1. Ordnung auch

$$\frac{|x^{(k)} - x^{(k+1)}|}{|x^{(k)}|} \leq \epsilon$$

als Abbruchbedingung.

1.2.3 Die Sekantenmethode

Als Modifikation des Regula-Falsi-Verfahrens verzichtet man bei der Sekantenmethode darauf, die Lösung durch zwei Näherungswerte einzuschließen.

Herleitung des Algorithmus: Zu zwei vorgegebenen Näherungswerten $x^{(0)}$ und $x^{(1)}$, welche die Lösung nicht notwendigerweise einzuschließen brauchen, bestimmt man $x^{(2)}$ als Nullstelle der Sekante

$$\beta = \frac{f(x^{(1)}) - f(x^{(0)})}{x^{(1)} - x^{(0)}}$$

zu $(x^{(0)}, f(x^{(0)}))$ und $(x^{(1)}, f(x^{(1)}))$. Ungeachtet der Vorzeichen bestimmt man aus $x^{(1)}, x^{(2)}$ eine nächste Näherung $x^{(3)}$.

Die Iterationsvorschrift der Sekantenmethode ergibt sich damit zu

$$x^{(k+1)} = x^{(k)} - f(x^{(k)})\frac{x^{(k)} - x^{(k-1)}}{f(x^{(k)}) - f(x^{(k-1)})}, \quad k = 1, 2, \ldots . \tag{1.4}$$

Als Algorithmus ergibt sich somit:

Algorithmus 1.2.3 (Sekantenverfahren) *Wähle $x^{(0)}, x^{(1)} \in \mathbb{R}$, $\epsilon \geq 0$.*
Für $k = 1, 2 \ldots$:

1) Berechne

$$x^{(k+1)} = x^{(k)} - f(x^{(k)}) \frac{x^{(k)} - x^{(k-1)}}{f(x^{(k)}) - f(x^{(k-1)})}.$$

2) Falls $|f(x^{(k+1)})| \leq \epsilon$, STOPP.

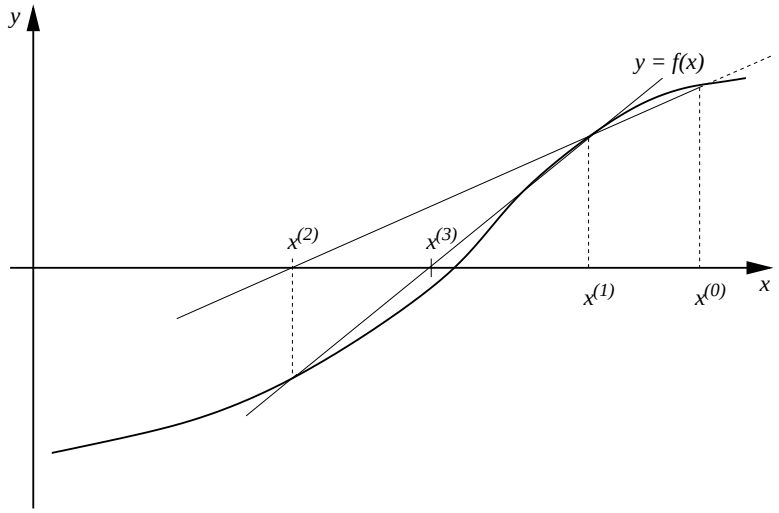


Abb. 1.4: Die geometrische Interpretation der Sekantenmethode.

Bemerkung 1.2.5 *Man beachte, dass das Sekantenverfahren im Gegensatz zu den vorangegangen Verfahren ein zweistufiges Verfahren ist.*

Satz 1.2.6 (Konvergenzordnung Sekantenverfahren) *Falls $f'(x^*) \cdot f''(x^*) \neq 0$ gilt, ist die Konvergenzordnung der Sekantenmethode $p = \frac{1}{2}(1 + \sqrt{5})$.*

Beweis. Wir betrachten die Sekantenmethode als Modifikation des Regula-Falsi-Verfahrens. Für hinreichend kleine Fehler $|\varepsilon_{k-1}|$ und $|\varepsilon_k|$ bleibt (1.3) für die Sekantenmethode gültig, bzw. geht über in

$$\varepsilon_{k+1} \approx \frac{f''(x^*)}{2f'(x^*)} \varepsilon_k \varepsilon_{k-1} \,. \tag{1.5}$$

Es besteht jedoch der Unterschied, dass bei Erhöhung von k sich beide Werte ε_{k-1} und ε_k ändern. Mit der Konstanten $C := |f''(x^*)/(2f'(x^*))|$ gilt für hinreichend großes k

$$|\varepsilon_{k+1}| \approx C|\varepsilon_k| \cdot |\varepsilon_{k-1}|.$$

Nach unserer Definition der Konvergenzordnung versuchen wir nun diese **Differenzengleichung** mit dem Ansatz

$$|\varepsilon_k| = \rho |\varepsilon_{k-1}|^p, \;\; \rho > 0, \; p \geq 1$$

zu lösen. Einsetzen ergibt

$$(|\varepsilon_{k+1}| =) \quad \rho|\varepsilon_k|^p = \rho \cdot \rho^p |\varepsilon_{k-1}|^{p^2} \stackrel{!}{=} C \cdot \rho |\varepsilon_{k-1}|^{p+1} \quad (= C|\varepsilon_k| \cdot |\varepsilon_{k-1}|) \,.$$

Diese letzte Gleichung kann aber für alle (hinreichend großen) k nur dann gelten, falls

$$\rho^p = C \text{ und } \; p^2 = p + 1$$

erfüllt sind. Die positive Lösung $p = \frac{1}{2}(1+\sqrt{5})$ der quadratischen Gleichung ist deshalb die Konvergenzordnung der Sekantenmethode. Die asymptotische Fehlerkonstante ist $\rho = C^{1/p} = C^{0.618}$. □

MATLAB-Funktion: SekantenMethode.m

```
function x1 = SekantenMethode(x0,x1,func,epsilon)
% Initialisierung
fx0 = func(x0); fx1 = func(x1);
while abs(fx1) > epsilon && abs(fx0-fx1) > epsilon
  tmp = x1;
  x1 = x1 - fx1 *(x1-x0)/(fx1-fx0);
  x0 = tmp;
  fx0 = fx1;
  fx1 = func(x1);
end
```

MATLAB-Beispiel:

Testen wir nun abschließend die Sekantenmethode an Bsp. 1.1.2 mit $K_0 = 10000$, $n = 48$ und $R = 250$.

```
>> n = 48;
>> K0 = 10000;
>> R = 250;
>> f = @(m) (m*K0-R)*(1+m)^n+R;
>> m = SekantenMethode(1e-2,0.1,f,1e-7)
m =
   0.00770147248822
```

Im Vergleich zu den oben gemachten Tests ist hier das Startintervall kleiner zu wählen.

1.2.4 Fixpunkt-Iteration

Idee: Wie schon bei klassischen Iterationsverfahren zur Lösung linearer Gleichungssysteme besteht die einfache Idee darin, das Nullstellen–Problem $f(x) = 0$, in ein Fixpunkt–Problem umzuschreiben. Wir werden schnell sehen, dass diese Idee auch für Systeme von nichtlinearen Gleichungen verwendet werden kann. Die Fixpunkt–Funktion lautet dann

$$\phi(x) := x - f(x).$$

Offensichtlich gilt $f(x^*) = 0$ genau dann, wenn $x^* = \phi(x^*)$, also wenn x^* Fixpunkt von ϕ ist. Nun sind auch andere Definitionen für ϕ denkbar, so dass wir eine ganze Klasse von Verfahren erhalten.

Die Konvergenz von Fixpunkt–Iterationen wird durch den *Banach*'schen[3] Fixpunktsatz geklärt. Wir formulieren und beweisen diesen hier in einem etwas allgemeineren Rahmen als er aus der Analysis bekannt sein dürfte. Insbesondere kann der Satz in dieser Formulierung auch im nächsten Abschnitt über nichtlineare Gleichungssysteme angewandt werden.

[3] Banach, Stefan (1892-1945)

Satz 1.2.7 ***(*Banach*'scher Fixpunktsatz)***
X sei ein linear normierter Raum mit Norm $\|\cdot\|$, $E \subseteq X$ sei eine vollständige Teilmenge von X. Die Abbildung $\Phi : X \to X$ erfülle folgende Bedingungen:

(i) ***Selbstabbildung:*** *$\Phi(E) \subseteq E$, also $\Phi : E \to E$.*

(ii) ***Kontraktion:*** *Es gelte $\|\Phi(x) - \Phi(y)\| \leq L\|x - y\| \quad \forall x, y \in E$ (*Lipschitz[4]*–Stetigkeit) mit einer Konstanten $L < 1$.*

Dann gilt:

(a) Es existiert genau ein Fixpunkt x^* von Φ in E.

(b) Die Iteration

$$x^{(k+1)} := \Phi(x^{(k)}), \quad k = 0, 1, 2, \ldots \tag{1.6}$$

konvergiert für jedes $x^{(0)} \in E$ gegen den Fixpunkt x^*.

(c) A-priori-Fehlerabschätzung:

$$\|x^{(k)} - x^*\| \leq \frac{L^k}{1 - L}\|x^{(1)} - x^{(0)}\|. \tag{1.7}$$

(d) A-posteriori-Fehlerabschätzung:

$$\|x^{(k)} - x^*\| \leq \frac{L}{1 - L}\|x^{(k)} - x^{(k-1)}\|. \tag{1.8}$$

Beweis. 1) Zeige, dass $(x^{(k)})_{k\in\mathbb{N}}$ eine *Cauchy*–Folge ist: Mit einer Teleskopsumme und der Dreiecks–Ungleichung gilt

$$\begin{aligned} \|x^{(k+m)} - x^{(k)}\| &= \|x^{(k+m)} - x^{(k+m-1)} + x^{(k+m-1)} + \ldots - x^{(k+1)} + x^{(k+1)} - x^{(k)}\| \\ &\leq \|x^{(k+m)} - x^{(k+m-1)}\| + \ldots + \|x^{(k+1)} - x^{(k)}\| \end{aligned}$$

sowie aufgrund der Kontraktivität:

$$\begin{aligned} \|x^{(k+1)} - x^{(k)}\| &= \|\Phi(x^{(k)}) - \Phi(x^{(k-1)})\| \\ &\leq L\|x^{(k)} - x^{(k-1)}\| \leq \ldots \leq L^k\|x^{(1)} - x^{(0)}\|, \end{aligned} \tag{1.9}$$

also wegen $L < 1$ und der (endlichen) geometrischen Reihe

$$\begin{aligned} \|x^{(k+m)} - x^{(k)}\| &\leq (L^{k+m-1} + \ldots + L^k)\|x^{(1)} - x^{(0)}\| \\ &= L^k\frac{1 - L^m}{1 - L}\|x^{(1)} - x^{(0)}\| \\ &\leq \frac{L^k}{1 - L}\|x^{(1)} - x^{(0)}\| \overset{k\to\infty}{\longrightarrow} 0. \end{aligned}$$

Also ist $(x^{(k)})_{k\in\mathbb{N}}$ eine *Cauchy*–Folge. Da E vollständig ist, existiert ein Grenzwert x^*, d.h.

$$\|x^{(k)} - x^*\| \overset{k\to\infty}{\longrightarrow} 0.$$

[4] Lipschitz, Rudolf (1832-1903)

2) Zeige, dass x^* ein Fixpunkt ist: Mit der Dreiecks–Ungleichung und der Iterationsvorschrift gilt

$$\begin{aligned}\|x^* - \Phi(x^*)\| &= \|x^* - x^{(k)} + x^{(k)} - \Phi(x^*)\| \\ &\leq \|x^* - x^{(k)}\| + \|\Phi(x^{(k-1)}) - \Phi(x^*)\| \\ &\leq \underbrace{\|x^* - x^{(k)}\|}_{\overset{k\to\infty}{\longrightarrow} 0} + L \underbrace{\|x^{(k-1)} - x^*\|}_{\overset{k\to\infty}{\longrightarrow} 0} \overset{k\to\infty}{\longrightarrow} 0,\end{aligned}$$

also $x^* = \Phi(x^*)$, da $k \in \mathbb{N}$ beliebig war.

3) **Eindeutigkeit:** Seien x^* und x^{**} zwei Fixpunkte, d.h.

$$x^* = \Phi(x^*) \text{ und } x^{**} = \Phi(x^{**}).$$

Dann gilt für $x^* \neq x^{**}$

$$\begin{aligned}\|\Phi(x^*) - \Phi(x^{**})\| &\leq L\|x^* - x^{**}\| \\ &= L\|\Phi(x^*) - \Phi(x^{**})\| \\ &< \|\Phi(x^*) - \Phi(x^{**})\|,\end{aligned}$$

da $L < 1$. Dies ist ein Widerspruch zu $x^* \neq x^{**}$, also folgt $x^* = x^{**}$.

4) **Fehlerabschätzung:** Für jedes $m > 0$ gilt wie in 1)

$$\begin{aligned}\|x^{(k)} - x^*\| &\leq \|x^{(k)} - x^{(k+1)}\| + \|x^{(k+1)} - x^{(k+2)}\| + \ldots + \|x^{(k+m)} - x^*\| \\ &= \|\Phi(x^{(k-1)}) - \Phi(x^{(k)})\| + \|\Phi(x^{(k)}) - \Phi(x^{(k+1)})\| \\ &\quad + \ldots + \|\Phi(x^{(k+m-1)}) - \Phi(x^*)\| \\ &\leq (L + L^2 + \ldots + L^m)\|x^{(k-1)} - x^{(k)}\| + L\|x^{(k+m-1)} - x^*\| \\ &= L\Bigg(\underbrace{\frac{1 - L^m}{1 - L}}_{\overset{m\to\infty}{\longrightarrow} \frac{1}{1-L}} \|x^{(k-1)} - x^{(k)}\| + \underbrace{\|x^{(k+m-1)} - x^*\|}_{\overset{m\to\infty}{\longrightarrow} 0}\Bigg)\end{aligned}$$

also (d) und mit (1.9) folgt dann (c).

□

Bemerkung 1.2.8 *1. Für $X = \mathbb{R}$, $\|\cdot\| \equiv |\cdot|$ (Betrag) und $E = [a, b]$ oder $E = \mathbb{R}$ erhält man den Satz für skalare Gleichungen. In diesem Fall bezeichnen wir die Fixpunkt-Funktion mit ϕ.*

2. Man beachte, dass der Satz insbesondere auch für unendlich-dimensionale Räume, wie z.B. Funktionenräume gilt.

Es stellt sich nun die Frage, wie man die Vorraussetzungen des *Banach*'schen Fixpunktsatzes nachprüfen bzw. nachrechnen kann. Die nachfolgende Folgerung aus dem *Banach*'schen Fixpunktsatz beantwortet dies.

Folgerung 1.2.9 *Sei $\phi \in C^1(\mathbb{R})$ mit $\phi : [a, b] \to [a, b]$ mit*

$$\max_{x\in[a,b]} |\phi'(x)| =: L < 1, \tag{1.10}$$

dann besitzt ϕ genau einen Fixpunkt und Satz 1.2.7 gilt für $\|\cdot\| \equiv |\cdot|$.

Beweis. Mit dem Mittelwertsatz gilt

$$|\phi(x) - \phi(y)| = |\phi'(\xi)(x-y)| \leq |x-y| \max_{\xi\in(a,b)} |\phi'(\xi)| = L|x-y|,$$

also ist ϕ eine Kontraktion. □

Für Fixpunkt-Iterationen kann man ein einfach zu handhabendes Kriterium herleiten, um die Konvergenzordnung zu bestimmen.

Lemma 1.2.10 (Konvergenzordnungskriterium für die Fixpunkt-Iteration) *Sei $\phi : \mathbb{R} \to \mathbb{R}$ $(p+1)$-mal stetig differenzierbar in einer Umgebung von $x^* = \phi(x^*)$ und es gelte*

$$\phi^{(j)}(x^*) = 0, \quad j = 1, \ldots, p-1, \quad \phi^{(p)}(x^*) \neq 0,$$

dann konvergiert $x^{(k+1)} := \phi(x^{(k)})$ lokal *genau von der Ordnung p.*

Bemerkung 1.2.11 (Lokale Konvergenz) *„Lokal" heißt wiederum für alle Startwerte $x^{(0)} \in \mathbb{R}$, die nahe genug bei x^* liegen! Zur Illustration des Begriffs „lokal"betrachten wir als Übung das Beispiel $f(x) = arctan(x) = 0$ und löse dies mit dem lokalen Newton-Verfahren, welches im nächsten Abschnitt behandelt wird.*

Beweis. Für $p = 1$ folgt die Behauptung aus Satz 1.2.7 (*Banach*'scher Fixpunktsatz).
Sei also $p > 1$: Wir entwickeln $\phi(x)$ um x^* und erhalten mit einem $\xi_k \in (x^{(k)}, x^*)$

$$\begin{aligned} x^{(k+1)} &= \phi(x^{(k)}) \\ &= \underbrace{\phi(x^*)}_{=x^*} + \sum_{j=1}^{p} \frac{(x^{(k)} - x^*)^j}{j!} \underbrace{\phi^{(j)}(x^*)}_{=0,\ j=1,\ldots,p-1} + \frac{(x^{(k)} - x^*)^{p+1}}{(p+1)!}\phi^{(p+1)}(\xi_k) \end{aligned}$$

also mit der Dreiecksungleichung

$$\frac{|x^{(k+1)} - x^*|}{|x^{(k)} - x^*|^p} \leq \frac{1}{p!}|\phi^{(p)}(x^*)| + \underbrace{\frac{|x^{(k)} - x^*|}{(p+1)!}|\phi^{(p+1)}(\xi_k)|}_{\substack{k\to\infty \\ \longrightarrow 0}},$$

somit folgt die Behauptung. □

Beispiel 1.2.12 (Einfluss der Formulierung der Fixpunkt-Funktion) Die Eigenschaften von ϕ (vor allem die Größe der Konstanten L) beeinflussen die Iteration maßgebich. Daher ist u.U. eine Umformung sinnvoll:

$$\begin{aligned} f(x) &:= 2x - \tan x, \qquad & \phi_1(x) &:= \tfrac{1}{2}\tan x, \\ & & \phi_2(x) &:= \arctan(2x). \end{aligned}$$

Beachte: ϕ_1 ist weder Selbstabbildung auf $I = [0, \frac{\pi}{2}]$ noch Kontraktion! Man untersuche dies für ϕ_2.

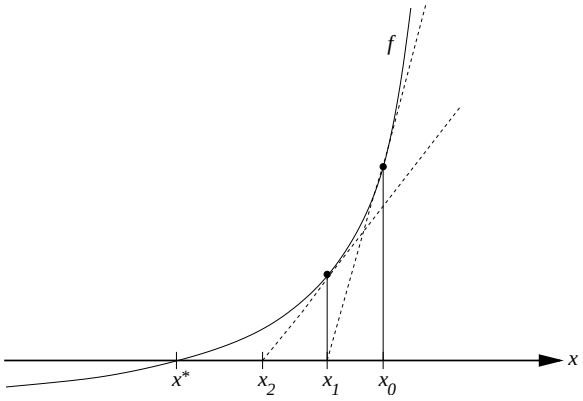


Abb. 1.5: Die geometrische Interpretation des *Newton*-Verfahrens.

1.2.5 Das Verfahren von *Newton*

Einerseits, da man das Sekantenverfahren aus dem vorangegangenen Unterabschitt aber auch als Vereinfachung des nun folgenden *Newton*[5]-Verfahren für skalare Funktionen betrachten kann, greifen wir hier schon mal vor und betrachten der Vollständigkeit halber zum Abschluss dieses Abschnitts über skalare nichtlineare Gleichungen noch das *Newton*-Verfahren im Eindimensionalen. Im nächsten Anschnitt wird dieses Verfahren, welches sich auch sehr gut zur Lösung nichtlinearer Gleichungssysteme eignet genauer behandelt.

Andererseits werden wir sehen, dass die Methode von *Newton* zur Klasse der Fixpunkt-Iterationen gehört.

Herleitung des Algorithmus: Ist die gegebene Funktion $f(x)$ der zu lösenden Gleichung $f(x) = 0$ stetig differenzierbar, so wird im Verfahren von *Newton* die Funktion $f(x)$ im Näherungswert $x^{(k)}$ linearisiert und der iterierte Wert $x^{(k+1)}$ als Nullstelle der Tangente in $x^{(k)}$ definiert (vgl. Abbildung 1.5). Mit anderen Worten, kann man das *Newton*-Verfahren für skalare Gleichungen geometrisch motivieren: der Schnittpunkt der Tangente mit der x-Achse ist die neue Näherung $x^{(k+1)}$. Aus der Tangentengleichung

$$T(x) = f(x^{(k)}) + (x - x^{(k)})f'(x^{(k)})$$

ergibt sich die Iterationsvorschrift

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})}\,. \tag{1.11}$$

Somit erhalten wir den Algorithmus

Algorithmus 1.2.4 (Newton-Verfahren) *Wähle* $x^{(0)} \in \mathbb{R}$, $\epsilon \geq 0$.
Für $k = 0, 1, 2 \ldots$*:*

1) Falls $|f(x^{(k)})| \leq \epsilon$*, STOPP.*

2) Berechne

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})}\,. \tag{1.12}$$

[5]Newton, Isaac (1642-1727)

Im nun folgenden Satz verwenden wir, dass die Methode von *Newton* zur Klasse der Fixpunkt-Iterationen mit der Funktion

$$\phi(x) := x - \frac{f(x)}{f'(x)} \quad \text{mit} \quad \phi(x^*) = x^*. \tag{1.13}$$

gehört, um die lokale quadratische Konvergenz des *Newton*-Verfahrens zu zeigen.

Satz 1.2.13 (Lokale Konvergenz des *Newton*-Verfahrens für skalare Probleme) *Es sei $I := [a, b]$ ein echtes Intervall mit $a < x^* < b$ und $f \in C^3(I)$ mit $f'(x^*) \neq 0$, d.h. x^* ist eine einfache Nullstelle von $f(x)$. Dann existiert ein Intervall $I_\delta = [x^* - \delta, x^* + \delta]$ mit $\delta > 0$, für welches die Fixpunkt-Funktion $\phi : I_\delta \to I_\delta$ definiert gemäß (1.13) eine Kontraktion darstellt. Ferner ist für jeden Startwert $x^{(0)} \in I_\delta$ die Konvergenz der Folge $(x_k)_{k\in\mathbb{N}}$ des* Newton-*Verfahrens mindestens von quadratischer Ordnung.*

Beweis. Für die erste Ableitung von ϕ erhalten wir

$$\phi'(x) = 1 - \frac{f'(x)^2 - f(x)f''(x)}{f'(x)^2} = \frac{f(x)f''(x)}{f'(x)^2}.$$

Da $f(x^*) = 0$, $f'(x^*) \neq 0$ und $f \in C^2(I)$ vorausgesetzt sind, gilt auch $\phi'(x^*) = 0$. Aus Stetigkeitsgründen existiert dann ein $\delta > 0$ derart, dass

$$|\phi'(x)| < 1 \quad \text{für alle } x \in [x^* - \delta, x^* + \delta] =: I_\delta$$

gilt. Somit ist ϕ eine Kontraktion in I_δ. Weiterhin sind für I_δ die Voraussetzungen des *Banach*'schen Fixpunktsatzes erfüllt und damit ist die Konvergenz von $(x_k)_{k\in\mathbb{N}}$ gezeigt.
Zum Beweis der Konvergenzordnung definieren wir $e^{(k+1)} := x^{(k+1)} - x^*$. Eine *Taylor*-Entwicklung von ϕ um x^* und $\phi'(x^*) = 0$ liefert

$$\begin{aligned} e^{(k+1)} &= x^{(k+1)} - x^* = \phi(x^{(k)}) - \phi(x^*) \\ &= \phi(x^* + e^{(k)}) - \phi(x^*) \\ &= \phi(x^*) + e^{(k)}\phi'(x^*) + \frac{(e^{(k)})^2}{2}\phi''(x^* + \theta_k e^{(k)}) - \phi(x^*) \quad \text{mit } \theta_k \in (0,1) \\ &= \frac{1}{2}(e^{(k)})^2\phi''(x^* + \theta_k e^{(k)}). \end{aligned}$$

Damit gilt also

$$\frac{|x^{(k+1)} - x^*|}{|x^{(k)} - x^*|^2} \leq \frac{1}{2} \sup_{0<\theta_k<1} |\phi''(x^* + \theta_k e^{(k)})| =: M_k\,.$$

Da aber auch

$$\phi''(x) = \frac{f'(x)^2 f''(x) + f(x)f'(x)f^{(3)}(x) - 2f(x)f''(x)^2}{f'(x)^3}$$

gilt und $f \in C^3(I)$ vorausgesetzt wurde, existiert ein $C \in \mathbb{R}$ mit $M_k \leq C$, für $k = 0, 1, 2, \ldots$ und somit ist das *Newton*-Verfahren quadratisch konvergent. □

MATLAB-Funktion: NewtonSimple.m

```
function [x,nit] = NewtonSimple(x,f,Df,tol,maxit,param)
nit = 0;
fx = f(x,param);
while norm(fx) > tol && nit <= maxit
  nit = nit+1;
  x = x-Df(x,param)\fx;
  fx = f(x,param);
end
nit
```

Beispiel 1.2.14 Vergleichen wir nun die Bisektionsmethode, das Sekantenverfahren und das *Newton*-Verfahren angewandt auf das Problem aus Bsp. 1.1.2 mit $K_0 = 10000$, $n = 48$ und $R = 250$ sowie Anfangsbedingungen $a = eps$, $b = 1$ für die Bisektionsmethode, $x^{(0)} = 1/2$, $x^{(1)} = 1$ für das Sekantenverfahren und $x^{(0)} = 0.1$ für das *Newton*-Verfahren, so erhalten wir die in Abb. 1.2.5 dargestellte Konvergenz.

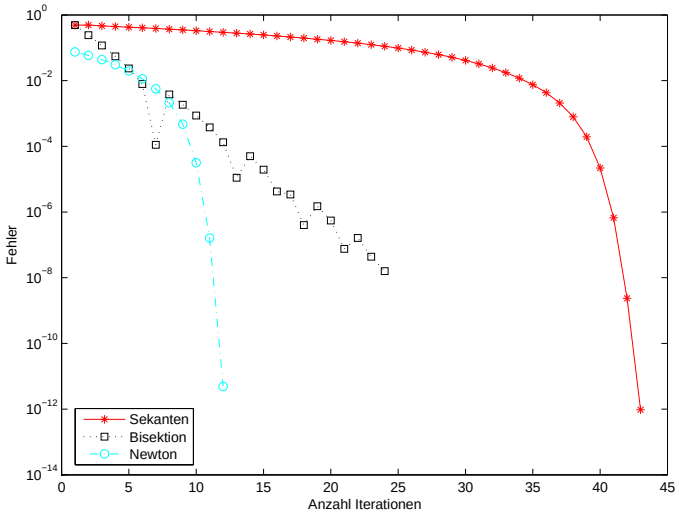


Abb. 1.6: Konvergenz von Bisektions-, Sekanten- und *Newton*-Verfahren angewandt auf $f(m) = (m \cdot K_0 - R) \cdot (1 + m)^n + R$ aus Bsp. 1.1.2 mit $K_0 = 10000$, $n = 48$ und $R = 250$.

1.2.6 Effizienz

Was nutzt eine hohe Konvergenzordnung, wenn man diese mit vielen Operationen „erkauft“? Ein solches Verfahren wäre ineffizient. Die „teuersten“ Operationen sind i.d.R. bei der Auswertung der Funktion f versteckt. Dies könnte ja z.B. die Lösung einer partiellen Differentialgleichung bedeuten. Daher verwendet man die Anzahl der Funktionsauswertungen als Synonym für den Aufwand bzw. die Anzahl der Operationen.

Definition 1.2.15 (Effizenzindex) *Sei m die Anzahl von Funktionsauswertungen pro Iterationsschritt. Damit definiert man den* ***Effizienzindex***

$$\text{ind} := p^{\frac{1}{m}}, \tag{1.14}$$

wobei p die Konvergenzordnung des Verfahrens angibt.

Beispiel 1.2.16 Wir fassen p und m für die oben vorgestellten Verfahren zusammen.

Verfahren	p	m	ind
Fixpunkt-Iteration	1	1	1
Newton	2	2^*	$\sqrt{2}$
Sekanten	$\frac{1+\sqrt{5}}{2}$	1	$\approx 1.6 (> \sqrt{2})$!
Regula Falsi	1	1	1

Unter der Annahme, dass die Auswertung von f in etwa so aufwendig ist wie die von f'.

Bemerkung 1.2.17 *Bislang haben wir neben der Stetigkeit von f — außer entsprechenden Differenzierbarkeit bei* Newton *— keine Eigenschaften von f vorausgesetzt. Für speziellere Funktionen kann man durch Ausnutzung der zusätzlichen Informationen effizientere Verfahren konstruieren. Eine Klasse „spezieller" Funktionen sind Polynome.*

1.2.7 Nullstellen von Polynomen

Auf den ersten Blick könnte man meinen, dass die Nullstellenbestimmung von Polynomen eine eher akademische Fragestellung wäre. Dem ist nicht so. Es gibt ganz konkrete Anwendungen z.B. bei der Bestimmung von Eigenwerten als Nullstelle des charakterisitschen Polynoms oder in der Computergraphik z.B. bei der Darstellung von Flächen und Bestimmung von Durchstosspunkten. Es stellt sich heraus, dass man für Polynome Varianten des *Newton*-Verfahrens konstruieren kann, die *global* konvergieren.

1.2.7.1 Globale Konvergenz des *Newton*–Verfahrens

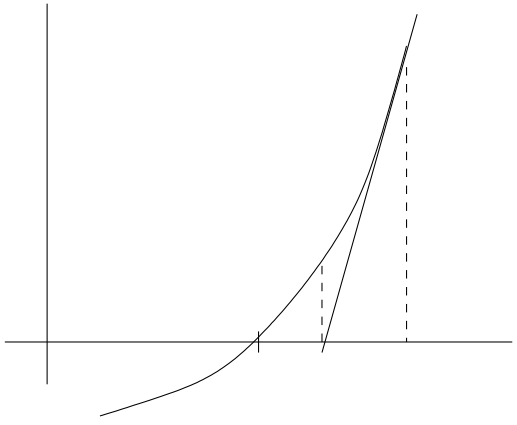
Für Polynome kann man unter bestimmten Voraussetzungen die *globale* Konvergenz des *Newton*-Verfahrens für Polynome beweisen. Grundlage ist der folgende Satz.

Satz 1.2.18 *Falls für $P \in \mathbb{P}_n$, $p(x^*) = 0$,*

$$P(x) > 0,\ P'(x) > 0,\ P''(x) \geq 0 \quad \forall x > x^*, \tag{1.15}$$

gilt, dann liefert das Newton*-Verfahren für alle Startwerte $x^{(0)} > x^*$ eine monoton fallende Folge, die gegen x^* konvergiert.*

Beweis. Der Beweis ist klar, wenn man z.B. die folgende Zeichnung betrachtet.



□

Bemerkung 1.2.19 *Aus Satz 1.2.18 bekommt man Schranken für x^* mit Hilfe der Koeffizienten von P.*

Folgerung 1.2.20 *Sei $P \in \mathbb{P}_n$ und x^* sei die betragsgrößte Nullstelle, $a_n = 1$. Dann gilt: Das* Newton–*Verfahren konvergiert für alle $x^{(0)}$ mit $|x^{(0)}| > |x^*|$ und sgn $(x^{(0)}) =$ sgn (x^*).*

Definition 1.2.21 (Begleitmatrix) *Sei $P(x) := \sum_{j=0}^{n} a_j x^j \in \mathbb{P}_n$, $a_n = 1$, dann heißt die Matrix*

$$A := \begin{bmatrix} & 0 & 1 & & & \\ & & \ddots & \ddots & & 0 \\ & 0 & & \ddots & \ddots & \\ & & & & 0 & 1 \\ -a_0 & \dots & \dots & & -a_{n-2} & -a_{n-1} \end{bmatrix} \in \mathbb{R}^{n\times n} \tag{1.16}$$

***Begleitmatrix** von p.*

Bemerkung 1.2.22 *Es gilt*

$$P_A(\lambda) = \det(\lambda I - A) = P(\lambda) = P_{A^T}(\lambda),$$

also ist p das charakteristische Polynom von A. Insbesondere sind also die Eigenwerte von A die Nullstellen von P. Dies können wir nun benutzen, um die gesuchten Schranken herzuleiten.

Sei $\|\cdot\|$ eine Norm auf dem $\mathbb{R}^n$ und $\|\cdot\|$ die induzierte Matrix–Norm, also

$$\|A\| = \sup_{x\neq 0} \frac{\|Ax\|}{\|x\|}.$$

Für einen Eigenwert λ^* von A zum Eigenvektor x^* gilt auf Grund der Submultiplikativität von $\|\cdot\|$

$$|\lambda^*|\,\|x^*\| = \|\lambda^* x^*\| = \|Ax^*\| \le \|A\|\,\|x^*\|,$$

d.h. wegen

$$|\lambda^*| \le \|A\| \tag{1.17}$$

ist die Matrix-Norm eine obere Schranke.

Beispiel 1.2.23 (a) Für die Zeilensummennorm $\|\cdot\|_\infty$ gilt also

$$\lambda^* \le \max\left\{1, \sum_{j=0}^{n-1} |a_j|\right\}. \tag{1.18}$$

(b) Für die Spaltensummennorm $\|\cdot\|_1$ gilt analog

$$\lambda^* \le \max\{|a_0|, 1+|a_1|, \dots, 1+|a_{n-1}|\}. \tag{1.19}$$

Diese Abschätzung kann man weiter verbessern, denn Eigenwerte sind invariant unter Ähnlichkeits–Transformationen: DAD^{-1}. Wähle dazu speziell

$$D := \begin{bmatrix} a_1 & & & 0 \\ & \ddots & & \\ & & a_{n-1} & \\ 0 & & & 1 \end{bmatrix},$$

also

$$DAD^{-1} = \begin{bmatrix} 0 & a_1/a_2 & & & 0 \\ \vdots & \ddots & \ddots & & \\ \vdots & & \ddots & a_{n-2}/a_{n-1} & \\ 0 & \dots & \dots & 0 & a_{n-1} \\ -a_0/a_1 & \dots & \dots & -a_{n-2}/a_{n-1} & -a_{n-1} \end{bmatrix}. \tag{1.20}$$

Mit $\|\cdot\|_1$ (Spaltensummennorm) erhält man also

$$\lambda^* \leq \max\left\{ \left|\frac{a_0}{a_1}\right|, 2\left|\frac{a_1}{a_2}\right|, \dots, 2\left|\frac{a_{n-2}}{a_{n-1}}\right|, 2|a_{n-1}| \right\} \tag{1.21}$$

und mit $\|\cdot\|_\infty$ (Zeilensummennorm)

$$\lambda^* \leq \left|\frac{a_0}{a_1}\right| + \left|\frac{a_1}{a_2}\right| + \dots + \left|\frac{a_{n-2}}{a_{n-1}}\right| + |a_{n-1}|. \tag{1.22}$$

1.2.7.2 Berechnung weiterer Nullstellen und Mackey–Trick

Mit dem obigen Verfahren kann man die betragsgrößte Nullstelle berechnen. Was macht man, wenn man z.B. alle Nullstellen benötigt? Sei t_1 die betragsgrößte Nullstelle von P, dann gilt

$$P(x) = (x - t_1)P_1(x)$$

mit $P_1 \in \mathbb{P}_{n-1}$, da P entsprechend faktorisiert werden kann. Man könnte nun P_1 berechnen, die betragsgrößte Nullstelle von P_1 bestimmen und dieses Vorgehen iterieren. Dies führt allerdings auf erhebliche Probleme:

- Rundungsfehler verstärken sich durch die Faktorisierung enorm. Man kann dann die Approximationen $(t_1, \dots, t_n)$ als Starwerte für eine neue Iteration („*Nachiteration*“) verwenden, der Aufwand ist aber groß.
- Man muss die Abspaltung explizit berechnen (die Polynomdivision führt zu einem hohen Aufwand).

Es gibt aber einen einfachen, aber sehr wirksamen Ausweg, den **Mackey–Trick:** Seien $t_1, \dots, t_r$ die r betragsgrößten Nullstellen von P, dann gilt:

$$P_r(x) = P(x)\left(\prod_{j=1}^{r}(x - t_j)\right)^{-1} \quad \text{für } x \neq t_j.$$

Für das *Newton*-Verfahren brauchen wir $P_r'(x)$. Hierfür gilt mit der Kettenregel:

$$\begin{aligned} P_r'(x) &= P'(x)\left(\prod_{j=1}^{r}(x - t_j)\right)^{-1} + P(x)\sum_{j=1}^{r}\frac{(-1)}{(x-t_j)^2}\prod_{\substack{i=1\\ i\neq j}}^{r}(x-t_i)^{-1} \\ &= \left(\prod_{j=1}^{r}(x - t_j)\right)^{-1}\left\{P'(x) - P(x)\sum_{j=1}^{r}\frac{1}{x-t_j}\right\}, \quad x \neq t_j. \end{aligned}$$

Also gilt für $x \neq t_j$ für den benötigten Quotienten:

$$\frac{P_r(x)}{P_r'(x)} = \frac{P(x)}{P'(x) - P(x)\sum\limits_{j=1}^{r}(x - t_j)^{-1}}, \tag{1.23}$$

d.h., die *Newton*–Korrektur

$$x^{(k+1)} = x^{(k)} - \frac{P_r(x^{(k)})}{P_r'(x^{(k)})}$$

kann leicht berechnet werden, *ohne* P_r oder P_r' explizit zu berechnen!

Allerdings: Das Auslöschungs–Problem hat man immer noch. Ein Ausweg sind sogenannte *„Sturm'sche Ketten"* (**Übung**).

1.3 Nichtlineare Gleichungssysteme

In diesem Abschnitt beschäftigen wir uns mit dem Lösen nichtlinearer Gleichungssysteme

$$F(x) = 0,$$

wobei $\Omega \subset \mathbb{R}^n$ und $F : \Omega \to \mathbb{R}^n$ eine stetige Funktion ist.

1.3.1 Fixpunkt–Iteration

Analog zum eindimensionalen Fall wandeln wir das Nullstellen-Problem $F(x) = 0$ in ein Fixpunkt-Problem mit Fixpunkt-Funktion

$$\Phi(x) := x - F(x)$$

um.

Folgerung 1.3.1 *Sei $\Omega \subseteq \mathbb{R}^n$ abgeschlossen und konvex, $\Phi : \Omega \to \mathbb{R}^n$ stetig differenzierbar mit $\Phi : \Omega \to \Omega$. Falls für eine beliebige Vektornorm $\|\cdot\|$ auf $\mathbb{R}^n$ und die zugehörige Matrixnorm*

$$\max_{x\in\Omega} \|J_\Phi(x)\| = L < 1 \tag{1.24}$$

gilt (J_Φ ist die Jacobi*–Matrix von Φ), dann gelten die Aussagen von Satz 1.2.7.*

Bemerkung 1.3.2 (Erzielen einer gewünschte Genauigkeit via a priori Fehlerabschätzung)
Der Banach*'sche Fixpunktsatz 1.2.7 sagt auch aus, wie viele Iterationen man machen muss, um eine gewünschte Genauigkeit zu erzielen.*
Ziel:

$$\|x^{(k)} - x^*\| \le \varepsilon$$

mit einer Toleranz $\varepsilon > 0$. Wegen der a priori Fehlerabschätzung (c) ist dies erfüllt, falls

$$\frac{L^k}{1-L}\|x^{(1)} - x^{(0)}\| \le \varepsilon,$$

also wegen $L < 1$

$$L^{-k} \ge \frac{\|x^{(1)} - x^{(0)}\|}{\varepsilon(1-L)},$$

und damit

$$k \ge \log\left(\frac{\|x^{(1)} - x^{(0)}\|}{\varepsilon(1-L)}\right) \Big/ \log(L^{-1}). \tag{1.25}$$

Beispiel 1.3.3 Betrachte das System

$$6x = \cos x + 2y, \quad 8y = xy^2 + \sin y \tag{1.26}$$

auf $\Omega = [0,1] \times [0,1]$. Gesucht sind also $x, y \in [0,1]$ mit (1.26). Definiere also die Fixpunkt-Funktion $\Phi : \mathbb{R}^2 \to \mathbb{R}^2$ durch

$$\Phi(x,y) := \begin{pmatrix} \frac{1}{6}\cos x + \frac{1}{3}y \\ \frac{1}{8}xy^2 + \frac{1}{8}\sin y \end{pmatrix}.$$

Für diese Funktion untersuchen wir die Voraussetzungen des *Banach*'schen Fixpunktsatzes.

- **Selbstabbildung:** Wegen $0 \le \cos x \le 1$, $0 \le \sin x \le 1 \ \ \forall x, y \in [0,1] \ \ (1 < \frac{\pi}{2})$ gilt also

$$\begin{aligned} 0 &\le \tfrac{1}{6}\cos x + \tfrac{1}{3}y &&\le \tfrac{1}{6} + \tfrac{1}{3} &&= \tfrac{1}{2} < 1 \\ 0 &\le \tfrac{1}{8}xy^2 + \tfrac{1}{8}\sin y &&\le \tfrac{1}{8} + \tfrac{1}{8} &&= \tfrac{1}{4} < 1 \end{aligned}$$

  Also gilt $\Phi : \Omega \to \Omega$, Φ ist also eine Selbstabbildung.

- **Kontraktion:** Wähle $\|\cdot\|_\infty$ (die induzierte Matrixnorm ist die Zeilensummennorm). Wegen

$$J_\Phi(x,y) = \begin{pmatrix} -\frac{1}{6}\sin x & \frac{1}{3} \\ \frac{1}{8}y^2 & \frac{1}{4}xy + \frac{1}{8}\cos y \end{pmatrix}$$

  gilt

$$\|J_\Phi\|_\infty = \max\left\{ \underbrace{\frac{1}{6}|\sin x| + \frac{1}{3}}_{\le \frac{1}{2}}, \underbrace{\frac{1}{8}y^2 + \frac{1}{4}xy + \frac{1}{8}\cos y}_{\le \frac{1}{2}} \right\} \le \frac{1}{2} =: L.$$

Somit erfüllt die Fixpunkt-Funktion die Vorraussetzungen des *Banach*'schen Fixpunktsatzes und wir können diesen anwenden. Wähle z.B. $(x_0, y_0) = (0,0)$. Angenommen, wir wollten die Lösung bis auf einen Fehler von $\varepsilon = 0.1$ bestimmen: $(x_1, y_1) = \Phi(x_0, y_0) = (\frac{1}{6}, 0)$

$$\begin{aligned} (x_2, y_2) &= \Phi(x_1, y_1) = \Phi\left(\frac{1}{6}, 0\right) \\ &= \left(\frac{1}{6}\cos\left(\frac{1}{6}\right), 0\right) \doteq (0.164, 0). \end{aligned}$$

Mit der Abschätzung (1.25) gilt mit $\varepsilon = 0.1$, $L = \frac{1}{2}$ und $\|(x_1, y_1) - (x_0, y_0)\|_\infty = \frac{1}{6}$

$$n \ge \log\Bigg(\underbrace{\frac{\frac{1}{6}}{\frac{1}{10}\cdot\frac{1}{2}}}_{=\frac{20}{6}=\frac{10}{3}} \Bigg) / \log(2) = \frac{\log 10 - \log 3}{\log 2} \doteq 1,74,$$

also reichen obige zwei Schritte aus, um die gewünschte Genauigkeit zu erzielen.

1.3.2 Newton–Verfahren für Systeme

Wir wollen nun analog zum eindimensionalen Fall das Nullstellen-Problem

$$F(x) = 0$$

für Funktionen

$$F : \mathbb{R}^n \to \mathbb{R}^n, n > 1, \quad F \in C^2(\mathbb{R}^n) \tag{1.27}$$

mit dem *Newton*–Verfahren lösen. Man beachte, dass die erste Ableitung der vektorwertigen Abbildung F hier eine *Matrix* ist. Die *Jacobi*–Matrix oder Funktionalmatrix ist gegeben durch

$$J_F(x) := \left(\frac{\partial F_i}{\partial x_j}(x)\right)_{i,j=1}^{n} = \begin{pmatrix} \frac{\partial F_1}{\partial x_1}(x) & \dots & \frac{\partial F_1}{\partial x_n}(x) \\ \vdots & & \vdots \\ \frac{\partial F_n}{\partial x_1}(x) & \dots & \frac{\partial F_n}{\partial x_n}(x) \end{pmatrix} \in \mathbb{R}^{n\times n}.$$

Also macht ein Ausdruck der Form

$$\frac{F(x)}{J_F(x)}$$

keinen Sinn!

Herleitung des Algorithmus: Die Iterierte $x^{(k)}$ sei bereits berechnet. Die Idee ist es, $F(x)$ durch *Taylor*-Entwicklung am Entwicklungspunkt $x^{(k)}$ linear zu approximieren. Mit dem Satz von Taylor gilt

$$F(x) = F(x^{(k)}) + J_F(x^{(k)})(x - x^{(k)}) + \mathcal{O}(\|x - x^{(k)}\|_2^2).$$

Wie in Abschnitt 1.2.5 bestimmt man in der k-ten Iteration also anstelle der Nullstelle von F die Nullstelle $x^{(k+1)}$ des Taylorpolynoms ersten Grades

$$T(x) = F(x^{(k)}) + J_F(x^{(k)})(x - x^{(k)}).$$

Falls die Matrix J_F in $x^{(k)}$ regulär (invertierbar) ist, folgt

$$T(x^{(k+1)}) = 0 \quad \Longleftrightarrow \quad x^{(k+1)} = x^{(k)} - (J_F(x^{(k)}))^{-1} F(x^{(k)}).$$

Formal müssten wir also die inverse Matrix $(J_F(x^k))^{-1}$ berechnen, was natürlich numerisch völlig unbrauchbar ist. Man kann dies aber leicht umgehen. Wir müssen in der k-ten Iteration ein lineares $(n \times n)$-Gleichungssystem der Form

$$J_F(x^{(k)})x^{(k+1)} = J_F(x^{(k)})x^{(k)} - F(x^{(k)})$$

mit Koeffizientenmatrix $J_F(x^{(k)})$ lösen. Da dies aufwendiger ist als eine einfache Fixpunkt-Iteration, reduziert man den Aufwand um eine Matrix-Vektor-Multiplikation durch eine *Newton*-Korrektur-Iteration

$$J_F(x^{(k)})s^{(k)} = -F(x^{(k)}), \quad x^{(k+1)} = x^{(k)} + s^{(k)}.$$

Somit erhalten wir den Algorithmus:

Algorithmus 1.3.1 (Newton–Iteration) *Sei $x^{(0)} \in \mathbb{R}^n$ ein "geeigneter" Starwert.*
Für $k = 0, 1, 2, \ldots$:

1) Bestimme einen Newton-*Schritt $s^{(k)} \in \mathbb{R}^n$ durch*

$$J_F(x^{(k)})s^{(k)} = -F(x^{(k)}) \tag{1.28}$$

*2) **Newton–Korrektur**: $x^{(k+1)} = x^{(k)} + s^{(k)}$*

Bemerkung 1.3.4 *In 1.) hat man also ein lineares Gleichungssystem zu lösen! Wir haben somit die numerische Lösung eines nichtlinearen Gleichungssystems auf die numerische Lösung einer Folge von linearen Gleichungssystemen übertragen. Je nach Größe von n macht man dies iterativ oder mittels LR/QR-Zerlegung (vgl. Numerik 1).*

Bemerkung 1.3.5 (Invarianzeigenschaft) *Sei $A \in \mathbb{R}^{n\times n}$ einer reguläre Matrix. Offensichtlich ist das Problem der Lösung von $F(x) = 0$ äquivalent zu dem Problem*

$$G(x) := AF(x) = 0.$$

Zugleich gilt auch

$$\begin{aligned} J_G(x)^{-1}G(x) &= (AJ_F(x))^{-1}AF(x) = J_F^{-1}(x)A^{-1}AF(x) \\ &= J_F^{-1}(x)F(x), \end{aligned}$$

d.h. sowohl das zu lösende Problem $F(x) = 0$ als auch das Newton*-Verfahren sind affin-invariant.*

1.3.2.1 Schnelle lokale Konvergenz des *Newton*–Verfahrens

Es stellt sich natürlich die Frage, ob man die lokal quadratische Konvergenz des *Newton*-Verfahrens für skalier Gleichungen auch bei Systemen erhält. Die Antwort ist „ja“, falls folgende Voraussetzung erfüllt ist.

Voraussetzung 1.3.6 *Sei $\Omega \subset \mathbb{R}^n$ offen und konvex, $F : \Omega \to \mathbb{R}^n$, $F \in C^1(\Omega)$, $J_F(x)$ regulär für alle $x \in \Omega$ und es gelte*

$$\|(J_F(x))^{-1}\| \le \beta \quad \forall x \in \Omega, \tag{1.29}$$

Weiterhin sei J_F auf Ω Lipschitz*-stetig mit Konstante γ, d.h.,*

$$\|J_F(x) - J_F(y)\| \le \gamma\|x - y\| \quad \forall x, y \in \Omega \tag{1.30}$$

und es existiere $x^* \in \Omega$ mit $F(x^*) = 0$.

Lemma 1.3.7 *Unter Voraussetzung 1.3.6 gilt*

$$\|F(x) - F(y) - J_F(y)(x - y)\| \le \frac{\gamma}{2}\|x - y\|^2 \quad \forall x, y \in \Omega. \tag{1.31}$$

Beweis. Sei $\varphi(t) := F(y + t(x - y))$, $t \in [0, 1]$. Dann gilt $\varphi \in C^1([0, 1]) \quad \forall x, y \in \Omega$ und mit der Kettenregel gilt

$$\varphi'(t) = J_F(y + t(x - y))(x - y).$$

Dann gilt

$$\begin{aligned} \|\varphi'(t) - \varphi'(0)\| &= \|[J_F(y + t(x - y)) - J_F(y)](x - y)\| \\ &\le \|J_F(y + t(x - y)) - J_F(y)\| \cdot \|(x - y)\| \\ &\overset{(1.30)}{\le} \gamma \cdot t\|x - y\|^2. \end{aligned}$$

Auf der anderen Seite gilt:

$$F(x) - F(y) - J_F(y)(x - y) = \varphi(1) - \varphi(0) - \varphi'(0) = \int_0^1 (\varphi'(t) - \varphi'(0))\, dt,$$

also

$$\begin{aligned} \|F(x) - F(y) - J_F(y)(x - y)\| &\le \int_0^1 \|\varphi'(t) - \varphi'(0)\|\, dt \\ &\le \gamma\|x - y\|^2 \underbrace{\int_0^1 t\, dt}_{=1/2} = \frac{\gamma}{2}\|x - y\|^2 \end{aligned}$$

und damit die Behauptung. □

Satz 1.3.8 (Lokale Konvergenz des *Newton*-Verfahrens) *Zusätzlich zu Voraussetzung 1.3.6 sei $x^{(0)} \in \Omega$ ein Starwert mit*

$$\|x^* - x^{(0)}\| < \omega \tag{1.32}$$

und

$$\frac{1}{2}(\beta\gamma\omega) < 1. \tag{1.33}$$

Dann bleibt die durch das Newton*–Verfahren definierte Folge $(x^{(k)})_{k\in\mathbb{N}}$ innerhalb der Kugel $K_\omega(x^*)$ und konvergiert (mindestens)* quadratisch *gegen x^*.*

Beweis. Zunächst gilt:

$$\begin{aligned} x^{(k+1)} - x^* &= x^{(k)} - x^* - J_F(x^{(k)}))^{-1}\Big(F(x^{(k)}) - \overbrace{F(x^*)}^{=0}\Big) \\ &= -(J_F(x^{(k)}))^{-1}\Big(F(x^{(k)}) - F(x^*) - J_F(x^{(k)})(x^{(k)} - x^*)\Big). \end{aligned}$$

Mit Lemma 1.3.7 folgt dann

$$\begin{aligned} \|x^{(k+1)} - x^*\| &\leq \underbrace{\|(J_F(x^{(k)}))^{-1}\|}_{\overset{(1.29)}{\leq}\beta} \underbrace{\|F(x^{(k)}) - F(x^*) - J_F(x^{(k)})(x^{(k)} - x^*)\|}_{\overset{(1.31)}{\leq}\frac{\gamma}{2}\|x^{(k)}-x^*\|^2} \\ &\leq \frac{\beta\gamma}{2}\|x^{(k)} - x^*\|^2, \end{aligned}$$

Wir erhalten also quadratische Konvergenz.

Zeige noch $(x^{(k)})_{k\in\mathbb{N}} \subset K_\omega$: für $k = 0$ ist dies gerade (1.32). Weiter induktiv:

$$\|x^{(k+1)} - x^*\| \leq \frac{\beta\gamma}{2}\|x^{(k)} - x^*\|^2 \overset{(I.A.)}{<} \frac{\beta\gamma}{2}\omega^2 \overset{(1.33)}{<} \omega.$$

□

Bemerkung 1.3.9 *(a) Falls $F \in C^2(\Omega)$, dann gilt (1.30) — wir haben also etwas weniger Regularität gefordert.*

(b) Die Konstanten β, γ sind durch das Problem gegeben. Dabei bedeutet (1.33), dass man gute Startwerte *benötigt. Dies sichert dann die* lokale *quadratische Konvergenz.*

(c) Unter obigen Voraussetzungen kann man zeigen, dass x^* die einzige Nullstelle in $K_{2/\beta\gamma}(x^*)$ ist.

Wir liefern nun einen Konvergenzsatz, der eine geringere Regularität von f voraussetzt.

Satz 1.3.10 *Sei $\Omega \subset \mathbb{R}^n$ offen und konvex, $F : \Omega \to \mathbb{R}^n$ eine stetig partiell differenzierbare Funktion mit invertierbarer* Jacobi*-Matrix $J_F(x)$ für alle $x \in \Omega$. Es gelte ferner für ein $\omega \geq 0$ die folgende* Lipschitz*-Bedingung:*

$$\|J_F^{-1}(x)(J_F(x + sv) - J_F(x))v\| \leq s\omega\|v\|^2$$

für alle $s \in [0, 1]$, $x \in \Omega$ und $v \in \mathbb{R}^n$ mit $x + v \in \Omega$. Weiterhin existiere eine Lösung $x^* \in \Omega$ und ein Startwert $x^{(0)} \in \Omega$ derart, dass

$$\rho := \|x^* - x^{(0)}\| < \frac{2}{\omega} \quad \text{und} \quad K_\rho(x^*) \subseteq \Omega.$$

Dann bleibt die durch das Newton-*Verfahren definierte Folge* $(x^{(k)})_{k\in\mathbb{N}}$ *für* $k > 0$ *in der offenen Umgebung* $K_\rho(x^*)$ *und konvergiert gegen* x^**, d.h.*

$$\|x^{(k)} - x^*\| < \rho \text{ für } k > 0$$

und

$$\lim_{k\to\infty} x^{(k)} = x^*.$$

Die Konvergenzgeschwindigkeit läßt sich abschätzen durch

$$\|x^{(k+1)} - x^*\| \le \frac{\omega}{2}\|x^{(k)} - x^*\|^2 \text{ für } k = 0, 1, 2, \ldots$$

und darüber hinaus ist die Lösung x^* *eindeutig in* $K_{2/\omega}(x^*)$*.*

Beweis. Als ersten Schritt des Beweises leiten wir aus der *Lipschitz*-Bedingung für $x, y \in \Omega$ her, dass gilt:

$$\|J_F^{-1}(x)(F(y) - F(x) - J_F(x)(y - x))\| \le \frac{\omega}{2}\|y - x\|^2. \tag{1.34}$$

Aus $\frac{\partial}{\partial s}F_j(x+s(y-x)) = \nabla F_j(x+s(y-x))\cdot(y-x)$ folgt die *Lagrange*-Form des Mittelwertsatzes der Integralrechnung

$$\int_0^1 \Big(J_F(x + s(y - x)) - J_F(x)\Big)(y - x)\, ds = F(y) - F(x) - J_F(x)(y - x).$$

Da auch $J_F^{-1}(x)$ unabhängig von s ist, erhalten wir für die linke Seite von (1.34)

$$\begin{aligned}
&\Big\|J_F^{-1}(x)\Big(F(y) - F(x) - J_F(x)(y - x)\Big)\Big\| \\
&= \Big\|\int_0^1 J_F^{-1}(x)\Big(J_F(x + s(y - x)) - J(x)\Big)(y - x)\, ds\Big\| \\
&\le \int_0^1 \Big\|J_F^{-1}(x)\Big(J_F(x + s(y - x)) - J_F(x)\Big)(y - x)\Big\|\, ds \\
&\le \int_0^1 s\omega\|y - x\|^2\, ds = \frac{\omega}{2}\|y - x\|^2.
\end{aligned}$$

Nun erhalten wir für die Iterationsvorschrift

$$x^{(k+1)} = x^{(k)} - J_F^{-1}(x^{(k)})F(x^{(k)})$$

sowie $F(x^*) = 0$

$$\begin{aligned}
x^{(k+1)} - x^* &= x^{(k)} - J_F^{-1}(x^{(k)})F(x^{(k)}) - x^* \\
&= x^{(k)} - x^* - J_F^{-1}(x^{(k)})(F(x^{(k)}) - F(x^*)) \\
&= J_F^{-1}(x^{(k)})\Big[f(x^*) - F(x^{(k)}) - J_F(x^{(k)})(x^* - x^{(k)})\Big].
\end{aligned}$$

Mit (1.34) erhalten wir nun

$$\|x^{(k+1)} - x^*\| \le \frac{\omega}{2}\|x^{(k)} - x^*\|^2.$$

Da $\|x^{(0)} - x^*\| = \rho$, folgt daraus

$$\|x^{(1)} - x^*\| \le \underbrace{\frac{\omega}{2}\|x^{(0)} - x^*\|}_{=\frac{\omega\rho}{2}=:\,\alpha<1}\|x^{(0)} - x^*\| < \|x^{(0)} - x^*\|$$

und somit induktiv für $k > 0$

$$\|x^{(k)} - x^*\| \leq \underbrace{\frac{\omega}{2}\|x^{(k-1)} - x^*\|}_{\leq \frac{\omega\rho}{2} = \alpha < 1} \|x^{(k-1)} - x^*\| \leq \alpha^k \|x^{(0)} - x^*\| < \|x^{(0)} - x^*\| \quad (k > 0)\,.$$

Daraus folgt $\|x^{(k)} - x^*\| < \rho$ für alle $k > 0$ und die Folge $(x^{(k)})_{k\in\mathbb{N}}$ konvergiert gegen x^*.

Zum Beweis der Eindeutigkeit in der Umgebung $K_{2/\omega}(x^*)$ um x^* mit Radius $2/\omega$ benutzen wir nochmals (1.34).
Sei $x^{**} \in K_{2/\omega}(x^*)$ eine weitere Lösung, also $f(x^{**}) = 0$ und $\|x^* - x^{**}\| < 2/\omega$.
Einsetzen in (1.34) liefert

$$\begin{aligned} \|x^{**} - x^*\| &= \|J_F^{-1}(x^*)\Big(0 - 0 - J_F^{-1}(x^*)(x^{**} - x^*)\Big)\| \\ &\leq \underbrace{\frac{\omega}{2}\|x^{**} - x^*\|}_{<1} \|x^{**} - x^*\|, \end{aligned}$$

dies ist aber nur möglich, falls $x^{**} = x^*$. □

Bemerkung 1.3.11 (Merkregel) *Das* Newton-*Verfahren konvergiert lokal quadratisch.*

Bemerkung 1.3.12 (Konvergenztest) *Als Näherung an den Fehler $\|x^{(k)} - x^*\|$ verwenden wir den Term $\|J_F^{-1}(x^{(k)})F(x^{(k)})\|$ (= $\|J_F^{-1}(x^{(k)})(F(x^{(k)}) - F(x^*))\|$). Da man erwartet, dass der Fehler monoton fällt, d.h. $\|x^{(k+1)} - x^*\| \leq \|x^{(k)} - x^*\|$ gilt, testet man dieses für jedes k durch den* **natürlichen Monotonietest***, d.h.*

$$\|J_F^{-1}(x^{(k)})F(x^{(k+1)})\| \leq \overline{\theta}\,\|J_F^{-1}(x^{(k)})F(x^{(k)})\| \quad \textit{sei erfüllt für ein } \overline{\theta} < 1. \tag{1.35}$$

Im Newton-*Verfahren berechnen wir zu $x^{(k)}$,*

$$J_F(x^{(k)})s^{(k)} = -F(x^{(k)}) \text{ und } x^{(k+1)} = x^{(k)} + s^{(k)}.$$

Somit ist der Ausdruck $J_F^{-1}(x^{(k)})F(x^{(k)})$ auf der rechten Seite in (1.35) *gleich dem Negativen der* Newton-*Korrektur $s^{(k)}$, die sowieso berechnet werden muss. Zusätzlich muss nur $\overline{s}^{(k)}$, definiert durch*

$$J_F(x^{(k)})\overline{s}^{(k)} = F(x^{(k+1)})\,,$$

bestimmt werden. Theoretische Untersuchungen und numerische Experimente liefern $\overline{\theta} = 1/2$ als eine gute Wahl. Falls der **natürliche Monotonietest***, d.h.*

$$\|\overline{s}^{(k)}\| \leq \frac{1}{2}\|s^{(k)}\|$$

für ein k verletzt ist, so ist das Newton-*Verfahren abzubrechen und es bleibt nichts Anderes übrig, als einen (hoffentlich) besseren Startwert zu finden.*

1.3.2.2 Hinweise zur Durchführbarkeit des *Newton*-Verfahrens und Varianten des Verfahrens

Wir betrachten nun einige Aspekte für die Realisierung des *Newton*-Verfahrens für Systeme.
Hinsichtlich der Durchführbarkeit des Newton-Verfahrens sind folgende drei Aspekte zu beachten:

- Oftmals kann man die *Jacobi*-Matrix $J_F(x^{(k)})$ nicht exakt oder nur mit riesigem Aufwand berechnen.

- Die Bestimmung des Newton-Schritts durch (1.28) bedeutet das Aufstellen und das Lösung eines linearen Gleichungssystems in *jeder* Iteration.

- Die Konvergenzresultate für das Newton-Verfahren sind lokaler Natur. D.h. sie gelten für Startwerte $x^{(0)}$ in der Nähe der gesuchten Nullstelle x^*. Das Newton-Verfahren ist *nicht* global konvergent.

Zunächst zum ersten Punkt:

Bemerkung 1.3.13 *(**Auswertung der** **Jacobi–Matrix durch numerische Approximation**) Das Problem, dass die* Jacobi-*Matrix $J_F(x^{(k)})$ nicht exakt oder nur mit riesigem Aufwand berechnet werden kann, kann durch folgende Auswege umgegangen werden:*

- *automatische Differenziation (AD), auch algorithmische Differenziation genannt,*

- *numerische Differenziation: Ersetzen der Ableitungen durch Differenzenquotienten.*

*Eine alternative Strategie liefert die Klasse der Quasi-*Newton*-Verfahren, welche im nächsten Abschnitt (vgl. Abschnitt 1.3.3) behandelt werden.*

Die anderen beiden Aspekte motivieren die folgenden beiden Varianten des Newton-Verfahrens.

Bemerkung 1.3.14 (Das vereinfachte *Newton*–Verfahren) *Um das Aufstellen und Lösen eines linearen Gleichungssystems in jeder Iteration zu vermeiden, wird hierbei in Gleichung (1.28) die* Jacobi-*Matrix $J_F(x^{(k)})$ durch $J_F(x^{(0)})$ zu ersetzt. Dadurch haben die Gleichungssysteme in jeder Iteration dieselbe Koeffizientenmatrix. Es reicht somit einmal eine QR- oder LR–Zerlegung von $J_F(x^{(0)})$ zu speichern. Allerdings geht aber die quadratische Konvergenz i.A. verloren.*

Bemerkung 1.3.15 (Wahl des Startwertes) *Wie bereits erwähnt kann Konvergenz oder Divergenz des Newton-Verfahrens sensibel von der Wahl des Startwertes $x^{(0)}$ abhängen!*
Man kann z.B. einige Schritte mit einem anderen Verfahren machen, um in den Konvergenzradius zu gelangen.

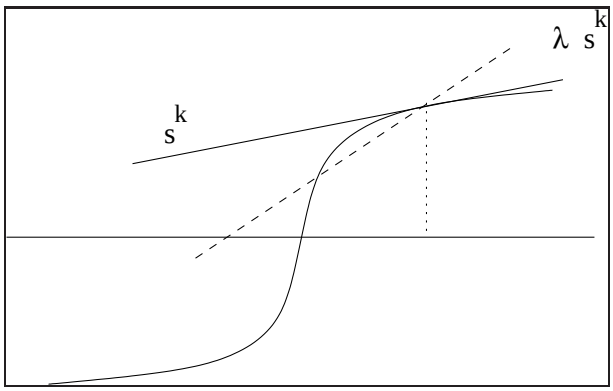
Gedämpftes *Newton*-Verfahren Eine Möglichkeit die Konvergenz des *Newton*-Verfahrens zu retten, ist häufig eine Dämpfung in der Form

$$x^{(k+1)} = x^{(k)} + \lambda_k s^{(k)}, \quad \text{für } \lambda_k \in (0, 1]\,.$$

Für eine einfache Dämpfungsstrategie können wir den Dämpfungsparameter λ_k derart wählen, so dass der natürliche Monotonietest für $\overline{\theta} = 1 - \lambda_k/2$ erfüllt ist, d.h.

$$\|J_F(x^{(k)})^{-1} F(x^{(k)} + \lambda_k s^{(k)})\| \leq \left(1 - \frac{\lambda_k}{2}\right) \|J_F(x^{(k)})^{-1} F(x^{(k)})\|\,.$$

Dabei wählen wir λ_k aus einer endlichen Folge $\{1, \frac{1}{2}, \frac{1}{4}, \ldots, \lambda_{\min}\}$ und brechen ggf. das Verfahren ab, falls $\lambda_k < \lambda_{\min}$ notwendig wäre. War λ_k erfolgreich, so zeigt die Praxis, dass es effizienter ist, mit $\lambda_{k+1} = \min\{1, 2\lambda_k\}$ fortzufahren anstatt wieder mit $\lambda_{k+1} = 1$ anzufangen. War der Monotonietest mit λ_k verletzt, so testet man erneut mit $\lambda_k/2$. Daraus ergibt sich der Folgende Algorithmus:



Algorithmus 1.3.2 (Gedämpftes *Newton*–Verfahren) *Wähle einen Startwert* $x^{(0)} \in \mathbb{R}^n$ *und* $\lambda_{min} > 0$. *Setze* $\lambda_0 = 1$
Für $k = 0, 1, 2, \ldots$

1.) Falls $\|F(x^{(k)})\| \leq \epsilon$, *STOPP.*

2.) Bestimme $s^{(k)} \in \mathbb{R}^n$ *durch Lösen von*

$$J_F(x^{(k)})s^{(k)} = -F(x^{(k)}).$$

3.) ***Dämfungsschritt:*** *Setze* $\lambda_k^{(0)} := \lambda_k$
.
Für $i = 0, 1, \ldots$ *:*

a) Berechne Testschritt $x^{(k+1,i)} := x^{(k)} + \lambda_k^{(i)} s^{(k)}$ *und prüfe*

$$\|J_F(x^{(k)})^{-1} F(x^{(k+1,i)})\| \leq \left(1 - \frac{\lambda_k^{(i)}}{2}\right) \|s^{(k)}\| \tag{1.36}$$

b) Falls der Monotonietest (1.36) erfüllt ist, akzeptiere Dämpfungsparameter gehe zu 4.), sonst

$$\lambda_k^{(i+1)} := \frac{\lambda_k^{(i)}}{2}.$$

c) Falls $\lambda_k^{(i+1)} < \lambda_{\min}$, *ABBRUCH.*

4.) ***Neuer Iterationsschritt:*** *Berechne*

$$x^{(k+1)} = x^{(k)} - \lambda_k^{(i)} s^{(k)}, \quad \textit{und} \quad \lambda_{k+1} = \min\{1, 2\lambda_k^{(i)}\}.$$

Bemerkung 1.3.16 (Globale Konvergenz des gedämpften *Newton*-Verfahrens)
Durch das Einführen der Dämpfung wird das ***Newton****-Verfahren globalisiert. D.h. der Bereich der Konvergenz wird erweitert. Allerdings konvergiert das gedämpfte* Newton*-Verfahren nur noch* linear.

MATLAB-Funktion: Newton.m

```
function [u,nit] = newton(u,F,DF,tol,maxit,param)
Fu = F(u,param);
DFu = DF(u,param);
s = -DFu\Fu;
lam = 1;
tmp = max(tol,tol*norm(s));
nit = 0;
while norm(s) > tmp && nit <= maxit
  nit = nit + 1;
  u_old = u;
  lam = min(1,2*lam);
  for k=1:30
    u = u_old + lam * s; % Daempfung mit Parameter lam
    Fu = F(u,param);
    if norm(DFu\Fu) <= (1-lam/2) * norm(s)
      break              % Abbruch der for-Schleife, falls
    end                  % Konvergenztest erfuellt
    lam = lam/2;         % lam noch zu groß --> halbieren
  end
  DFu = DF(u,param);
  s  = -DFu\Fu;
end
```

1.3.2.3 Beispiele: Das *Newton*-Verfahren für Optimierungsprobleme

Man kann das *Newton*–Verfahren auch zur Minimierung zweimal stetiger Funktionen $f : \mathbb{R}^n \to \mathbb{R}$ benutzen. Die zweimal stetige Differenzierbarkeit von f ergibt, dass der Gradient $\nabla f : \mathbb{R}^n \to \mathbb{R}^n$ stetig differenzierbar ist. Die notwendige Bedingung erster Ordnung

$$\nabla f(x) = 0$$

ist somit ein Gleichungssystem, auf welches wir das Newton–Verfahren anwenden können. Zur Illustration betrachten wir zwei Beispiele:

Beispiel 1.3.17 (Extremalstellen der *Rosenbrock*-Funktion) Es gilt für

$$f(x) = \sum_{i=1}^{n-1} \left[(1-x_i)^2 + 100(x_{i+1} - x_i^2)^2\right], \quad x \in \mathbb{R}^n$$

aus Beispiel 1.1.4, dass die *Jacobi*-Matrix $A := A(x) := J_f(x)$ folgende Einträge enthält

$$\begin{aligned} a_{11} &= 2 + 1200x_1^2 - 400x_2, \\ a_{jj} &= 202 + 1200x_j^2 - 400x_{j+1}, \quad j = 2, \ldots, n-1, \\ a_{nn} &= 200, \\ a_{j,j+1} &= a_{j+1,j} = -400x_j, \quad j = 1, \ldots, n-1, \\ a_{jk} &= 0, \quad |j-k| \geq 2\,. \end{aligned}$$

MATLAB-Funktion: rosenbrock.m

```
function value = rosenbrock(x,param)
n = size(x,1);
```

```
value(2:n,1) = 200 * (x(2:end)-x(1:end-1).^2);
value(1:n-1,1) = value(1:n-1,1) - 2 *(1-x(1:end-1)) ...
                 - 400*x(1:end-1).*(x(2:end)-x(1:end-1).^2);
```

MATLAB-Funktion: D_rosenbrock.m

```
function D = D_rosenbrock(x,param)
n = size(x,1);
d0(2:n,1) = 200;
d0(1:n-1) = d0(1:n-1) + 2 + 1200*x(1:n-1).^2-400*x(2:n);
dm1 = -400*x;
dp1 = -400*x([1,1:n-1]);
D = spdiags([dm1,d0,dp1],[-1 0 1],n,n);
```

MATLAB-Beispiel:

Man kann z.B. neben dem globalen Minimum $(x_1,\ldots,x_n)^T = (1,\ldots,1)^T$ mit dem *Newton*-Verfahren ein weiteres lokales Minimum in der Umgebung von $(x_1,x_2,\ldots,x_n)^T = (-1,1,\ldots,1)^T$ von f in Bsp. 1.1.4 finden.

```
>> n = 6;
>> x0 = [-1;ones(n-1,1)];
>> [x,nit] = NewtonSimple(x0,@rosenbrock,
   @D_rosenbrock,1e-10,100,[])
x =
  -0.98657497957099
   0.98339822883618
   0.97210667005309
   0.94743743682644
   0.89865118485173
   0.80757395203542
nit =
     5
```

Beispiel 1.3.18 (p-Laplace) Im Folgenden betrachten wir das p-**Laplace Problem**. Gegeben sei ein Gebiet $\Omega \subset \mathbb{R}^d$, $d \in \mathbb{N}$ und $1 < p < \infty$. Finde

$$u \in W_0^{1,p}(\Omega) := \{u \in L^p(\Omega) \,|\, \nabla u \in (L^p(\Omega))^d, u|_{\partial\Omega} = 0\}$$

mit

$$\begin{aligned} \operatorname{div}(|\nabla u|^{p-2}\nabla u) &= f \quad && \text{im Gebiet } \Omega, \\ u &= 0 && \text{auf dem Rand } \Gamma := \partial\Omega, \end{aligned} \tag{1.37}$$

wobei div den Divergenzoperator[6] bezeichne.
Die Lösung des p-Laplace-Problems (1.37) ist der Minimierer des Energiefunktionals

$$\mathcal{J}(u) := \int_\Omega \frac{1}{p}|\nabla u|^p - f u \, dx$$

[6] Für $u = (u_1,\ldots,u_d)^T \in C^1(\Omega;\mathbb{R}^d)$ sei $\operatorname{div}(u) := \sum_{i=1}^d \frac{\partial u_i}{\partial x_i}$.

über alle Funktionen u aus dem ***Sobolev***[7]**-Raum** $W_0^{1,p}(\Omega)$. Im Mehrdimensionalen ist dieses Problem im Allgemeinen nicht analytisch zu lösen. Für den Spezialfall $p = 2$, welches auf ein lineares Problem führt, spricht man einfach vom Laplace-Problem. Das p-Laplace-Problem ist ein typisches Beispiel für eine große Klasse von nichtlinearen Problemen.

Ohne jetzt auf die funktionalanalytischen Grundlagen einzugehen, beschränken wir uns kurzer Hand auf die Approximation von u durch eine stückweise lineare Funktion u_n und den eindimensionalen Fall ($d = 1$), um die Darstellung übersichtlich zu halten.

Gegeben sei $n \in \mathbb{N}$. Es sei $x_i = -1 + \frac{2i}{n+1}$ $(i = 1, \ldots, n)$, $h_i := x_i - x_{i-1}$ und

$$\varphi_i(x) := \begin{cases} \frac{x - x_{i-1}}{x_i - x_{i-1}} & \text{, falls } x \in [x_{i-1}, x_i], \\ \frac{x_{i+1} - x}{x_{i+1} - x_i} & \text{, falls } x \in (x_i, x_{i+1}], \\ 0 & \text{, sonst} \end{cases} \tag{1.38}$$

definiert sogenannte stückweise lineare **Hutfunktionen**. Man beachte, dass $\varphi_i(x_j) = \delta_{ij}$ gilt, wobei δ_{ij} das *Kronecker*-Symbol sei.

Wir suchen nun für $1 < p < \infty$ eine stückweise lineare Funktion $u_n(x) := \sum_{i=1}^n \alpha_i \varphi_i(x)$ (diese erfüllt per Definition die Randbedingungen $u_n(-1) = u_n(1) = 0$), welche zu gegebenem $f : [-1, 1] \to \mathbb{R}$ das Funktional

$$\widehat{\mathcal{J}}(u_n) = \mathcal{J}(\alpha) := \int_{-1}^{1} \frac{1}{p} |u_n'(x)|^p - f(x) u_n(x)\, dx$$

minimiert, mit $\alpha = (\alpha_1, \ldots \alpha_n)^T$. Man beachte $u_n'(x)|_{(x_{k-1}, x_k)} = h_k^{-1}(\alpha_{k-1} - \alpha_k)$ und

$$\begin{aligned} \mathcal{J}(\alpha) &:= \sum_{i=1}^{n} \sum_{k=0}^{n} \int_{x_k}^{x_{k+1}} \frac{1}{p} |\alpha_i \varphi_i(x)'|^p - \alpha_i f(x) \varphi_i(x)\, dx \\ &= \sum_{k=0}^{n} \int_{x_k}^{x_{k+1}} \frac{1}{p} |\alpha_k h_{k+1}^{-1} - \alpha_{k+1} h_{k+1}^{-1}|^p - f(x)(\alpha_k h_{k+1}^{-1} - \alpha_{k+1} h_{k+1}^{-1})\, dx\,. \end{aligned}$$

Ohne es zu beweisen, gehen wir davon aus, dass die gesuchte Lösung

$$\frac{\partial}{\partial \alpha_j} \mathcal{J}(\alpha) = 0, \quad j = 1, \ldots, n$$

erfüllt. Dies führt zu dem nichtlinearen Gleichungssystem

$$F(\alpha) = \big(F_1(\alpha), \ldots, F_N(\alpha)\big)^T = 0$$

mit

$$\begin{aligned} F_i(\alpha) &:= \frac{\partial}{\partial \alpha_i} \mathcal{J}(\alpha) \\ &= \frac{\partial}{\partial \alpha_i} \sum_{k=i-1}^{i} \int_{x_k}^{x_{k+1}} |\alpha_k h_{k+1}^{-1} - \alpha_{k+1} h_{k+1}^{-1}|^p - f(x)(\alpha_k h_{k+1}^{-1} - \alpha_{k+1} h_{k+1}^{-1})\, dx \\ &= -|\alpha_{i-1} h_i^{-1} - \alpha_i h_i^{-1}|^{p-2}(\alpha_{i-1} h_i^{-1} - \alpha_i h_i^{-1}) + h_i^{-1} \int_{x_{i-1}}^{x_i} f(x)\, dx \\ &\quad + |\alpha_i h_{i+1}^{-1} - \alpha_{i+1} h_{i+1}^{-1}|^{p-2}(\alpha_i h_{i+1}^{-1} - \alpha_{i+1} h_{i+1}^{-1}) - h_{i+1}^{-1} \int_{x_i}^{x_{i+1}} f(x)\, dx\,. \end{aligned}$$

[7] Sobolev, Sergei Lvovich (1908 - 1989)

Für $f \equiv 1$ erhalten wir

$$u(x) = \frac{p-1}{p}\left(1 - x^{\frac{p}{p-1}}\right) \quad 1 < p < \infty \tag{1.39}$$

bzw. für $p = 2$ gilt $u_n(x_i) = u(x_i)$, $i = 0, \ldots, n+1$, und für $p = 3/2$ lässt sich mit $n = 2M$

$$u(x) - u_n(x) = \frac{\frac{p-1}{p} - 2^{\frac{1}{1-p}}}{M^2}\left(1 - |x|^{(2-p)/(p-1)}\right) \tag{1.40}$$

zeigen.

Aufgabe 1.3.19 Man zeige (1.39) und (1.40).

Gilt $f \equiv 1$, so lässt sich $F(\alpha)$ exakt bestimmen, was in der Matlab-Routine `f2.m` realisiert ist. Die Berechnung der Funktionalmatrix $\frac{\partial^2}{\partial\alpha_i\partial\alpha_j}\mathcal{J}(\alpha)$ ist in der Matlab-Routine `Df2.m` umgesetzt.

MATLAB-Funktionen: f2.m und Df2.m

```
function Fu = f2(u,param)
% Aufstellen des Vektors Fu
h = 1/param.M;
u = [0;u;0];
Fu=zeros(2*param.M+1,1);
for j = 1:2*param.M
  stima = ([1 -1;-1 1]*u([j,j+1]))/h^2;
  fac = (u([j,j+1])'*stima)^((param.p-2)/2);
  Fu([j,j+1])=Fu([j,j+1])+h*fac*stima;
end
for j=1:2*param.M
  Fu([j;j+1])=Fu([j,j+1])-h/2;
end
Fu([1,end])=[];
```

```
function DFu = Df2(u,param)
% Aufstellen der Jacobimatrix
h = 1/param.M; u = [0;u;0];
DFu=sparse(2*param.M+1,2*param.M+1);
for j = 1:2*param.M
  stima = [1 -1;-1 1]/h^2;
  fac = (param.p-1)*(u([j,j+1])'*stima*u([j,j+1]))^((param.p-2)/2);
  DFu([j,j+1],[j,j+1])=DFu([j,j+1],[j,j+1])+h*fac*stima;
end
DFu = DFu(2:2*param.M,2:2*param.M);
```

MATLAB-Beispiel:

Man kann z.B. das p-Laplace-Problem näherungsweise mit dem folgenden Befehlen bestimmen.

```
>> M = 8;
>> p = 1.2;
>> u0 = (1-linspace(-1,1,2*M+1).^2)';
>> u0([1,end]) = [];
>> u = Newton(u0,@f2,@Df2,1e-12,...
          100,struct('M',M,'p',p));
>> xi = linspace(-1,1,2*M+1);
>> plot(xi,[0;u;0],'-*');
```

Für $p \to 1$ oder $p \to \infty$ nimmt die Nichtlinearität immer stärker zu. So liefert z.B. die Routine `NewtonSimple` für $p = 1.2$ und $M = 8$ schon keine Konvergenz mehr, jedoch die Routine `Newton` mit Schrittweitensteuerung und modifiziertem Abbruchkriterium.

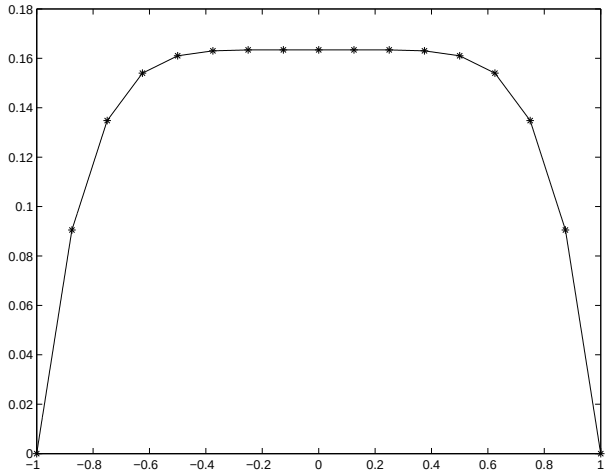


Abb. 1.7: Näherungsweise Lösung des p-Laplace-Problems mit $p = 1.2$ und $n = 15$.

1.3.3 Das *Broyden*-Verfahren

Das *Newton*-Verfahren, wie wir es bisher diskutiert haben, hat aufgrund seiner quadratischen Konvergenz eine große Bedeutung, besitzt aber auch einige Nachteile. Einer davon ist, dass in jeder Iteration die *Jacobi*-Matrix benötigt wird. In vielen Beispielen sind die analytischen Ableitungen jedoch nicht bekannt.

Das auf *Broyden*[8] zurückgehende Quasi-*Newton*-Verfahren ist ein Kompromiss zwischen Neuberechnung der *Jacobi*-Matrix in jedem Iterationsschritt (analog zum *Newton*-Verfahren) und der Verwendung einer festen Matrix im Lauf der gesamten Iteration. Ausgehend von einer Näherung an die *Jacobi*-Matrix werden die Matrizen in jedem Schritt so aktualisiert, dass sie möglichst ähnliche Abbildungseigenschaften aufweisen wie die exakte Funktionalmatrix. Gleichzeitig achtet man darauf, dass diese Aktualisierung möglichst wenig zusätzlichen Rechenaufwand erfordert. Sie erfolgt daher mittels Rang-1-Korrekturmatrizen, die sich aus Termen berechnen lassen, die ohnehin während der Iteration bestimmt werden müssen.
Die Näherungen an die *Jacobi*-Matrizen seien mit B_k bezeichnet. Dann wird im k-ten Schritt des Broyden-Verfahrens

$$B_k s^{(k)} = -F(x^{(k)}) \tag{1.41}$$

berechnet, wobei $x^{(k+1)} = x^{(k)} + s^{(k)}$ sei. Im Eindimensionalen liefert das Sekantenverfahren zu

[8] Broyden, Charles George (1933 -)

zwei gegebenen Werten $x^{(k-1)}$ und $x^{(k)}$ iterativ die Steigung der Sekante durch

$$\beta_k = \frac{f(x^{(k)}) - f(x^{(k-1)})}{x^{(k)} - x^{(k-1)}}, \quad k \geq 0 \tag{1.42}$$

und daraus eine weitere Näherung $x^{(k+1)}$ an x^* mittels

$$x^{(k+1)} = x^{(k)} - \beta_k^{-1} f(x^{(k)}) .$$

Die formale Verallgemeinerung der Sekanten-Bedingung (1.42) an B_k lautet

$$B_k s^{(k-1)} = F(x^{(k)}) - F(x^{(k-1)}) =: \delta F_k .$$

Bedingungen dieser Art werden **Quasi-Newton-Gleichung** genannt. Die Bedingung genügt jedoch nicht, um $B_k \in \mathbb{R}^{n\times n}$ eindeutig zu bestimmen. Daher versucht man für $k \geq n$ die letzten gewonnenen Informationen zu nutzen, so dass B_k, $k \geq n$ eine Lösung der folgenden Menge von n Systemen

$$B_k(x^{(k)} - x^{(k-j)}) = F(x^{(k)}) - F(x^{(k-j)}), \quad j = 1, \ldots, n \tag{1.43}$$

ist. Da im Allgemeinen die Vektoren $x^{(k-j)}, \ldots, x^{(k)}$ nicht linear unabhängig sind, fordern wir zusätzlich, dass die Differenz zwischen den linearen Approximationen von $F(x^{(k-1)})$ und $F(x^{(k)})$, nämlich

$$d_k := F(x^{(k)}) + B_k(x - x^{(k)}) - \left(F(x^{(k-1)}) + B_{k-1}(x - x^{(k-1)})\right) , \tag{1.44}$$

in der *Euklidischen* Norm minimiert wird. Setzen wir $j = 1$ in (1.43) so wird (1.44) zu

$$d_k = (B_k - B_{k-1})(x - x^{(k-1)}). \tag{1.45}$$

Zerlegt man den Vektor $x - x^{(k-1)}$ in der Form

$$x - x^{(k-1)} = \alpha\, s^{(k-1)} + r$$

mit $\alpha \in \mathbb{R}$ und $r^T s^{(k-1)} = 0$, dann erhält man aus (1.45)

$$d_k = \alpha(B_k - B_{k-1})s^{(k-1)} + (B_k - B_{k-1})r . \tag{1.46}$$

Da $(B_k - B_{k-1})s^{(k-1)} = \delta F_k - B_{k-1}s^{(k-1)}$ gilt, ist somit der erste Term in (1.46) unabhängig von B_k und es bleibt nur der zweite Term in (1.46) zu minimieren. Die Matrix B_k, die $(B_k - B_{k-1})r$ minimiert für alle r, die orthogonal zu $s^{(k-1)}$ sind, unter der Restriktion, dass (1.43) gilt, kann rekursiv mittels des **Rang-1-Updates** von B_{k-1}

$$B_k = B_{k-1} + \frac{\left(\delta F_k - B_{k-1}s^{(k-1)}\right)(s^{(k-1)})^T}{(s^{(k-1)})^T\, s^{(k-1)}} \tag{1.47}$$

berechnet werden. Die Methode (1.41) mit der Wahl (1.47) wird als Broyden-Verfahren bezeichnet. Zur Initialisierung setzt man $B_0 = J_F(x_0)$ oder eine geeignete Approximation z.B. mittels Differenzenquotienten, d.h. $(B_0)_{ij} \approx (F_i(x_0 + he_j) - F_i(x_0))/h$.

Bemerkung 1.3.20 *Ist eine QR-Zerlegung von $B_0 \in \mathbb{R}^{n\times n}$ gegeben, so läßt sich die QR-Zerlegung von B_k, $k > 0$ aus der QR-Zerlegung von B_{k-1} (wie wir in Numerik 1 gezeigt haben) mit $\mathcal{O}(n^2)$ bestimmen.*

MATLAB-Funktion: Broyden.m

```
function [x,nit] = Broyden(x,f,B,tol,maxit,param)
fx = f(x,param);
fx1 = zeros(size(fx));
nit = 0; err = inf;
while nit < maxit && err > tol
  s = - B\fx;
  x = x + s;
  err = norm(s);
  if err > tol
    fx1 = f(x,param);
    B = B + 1 /(s'*s) * fx1 * s';
  end
  fx = fx1;
  nit = nit +1;
end
```

Unter Verwendung des *Broyden*-Verfahrens lösen wir das nichtlineare Problem aus Bsp. 1.1.4 für $n = 6$. Diese Methode konvergiert in 18 Iterationen verglichen mit den 5 Iterationen, die das *Newton*-Verfahren erforderte bei gleichem Startwert $x^{(0)} = (-1, 1, \ldots, 1,)^T$. Die Matrix B_0 wurde gleich der *Jacobi*-Matrix im Punkt $x^{(0)}$ gesetzt. Abbildung 1.8 zeigt das Verhalten der *Euklidischen* Norm des Fehlers beider Methoden.

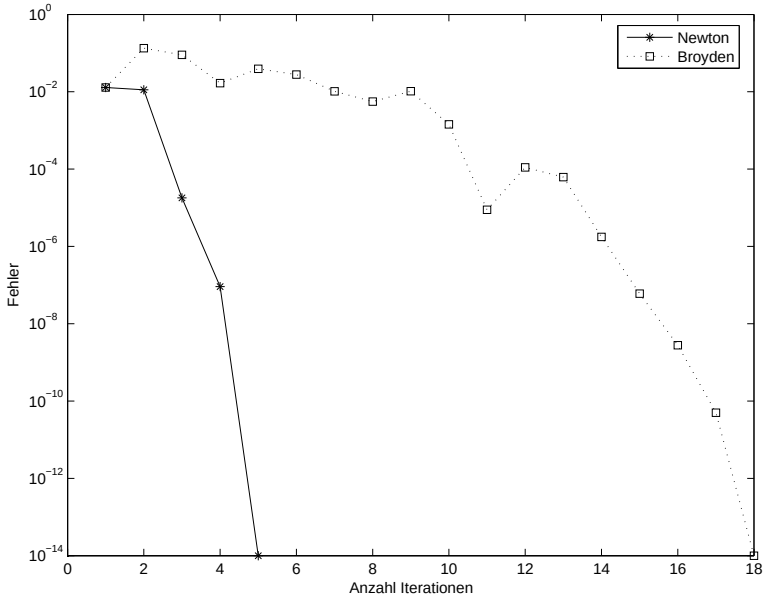


Abb. 1.8: Euklidische Norm des Fehlers für das *Broyden*- und *Newton*-Verfahren im Fall des nichtlinearen Problems aus Bsp. 1.1.4 für $n = 6$.

1.3.4 Gauß-Newton Verfahren für nichtlineare Ausgleichsprobleme

Zum Abschluss dieses Kapitels über nichtlineare Gleichungen betrachten wir hier noch mal Ausgleichsprobleme. In Numerik 1 hatten wir bereits lineare Ausgleichsprobleme behandelt. Zur Er-

innerung, bei linearen Ausgleichsproblemen hatten wir das Problem, bestimme $x \in \mathbb{R}^n$ mit

$$\begin{array}{cc} \|Ax- & b\| \to \min \\ \uparrow & \uparrow \\ \text{Parameter} & \text{Daten, Messungen} \end{array} \tag{1.48}$$

mit $A \in \mathbb{R}^{m\times n}, m \gg n$ und $b \in \mathbb{R}^m$ betrachtet.

Hierbei ist der Ausganspunkt eine Messung mit m Messpunkten $(t_i, b_i), i = 1, \ldots, m$. Diese Messungen unterliegen einem funktionaler Zusammenhang, welcher z.B. durch Modellannahmen gegeben ist. Dies ist die Modellfunktion y mit $b(t) = y(t, x_1, \ldots, x_n)$, in die n unbekannten Parameter $x_1, \ldots, x_n$ eingehen. Wären die Messungen exakt, d.h. ohne Messfehler, und würden exakt dem postulierten Modellzusammenhang folgen so hätte man $b_i = b(t_i) = y(t_i, x_1, \ldots, x_n), i = 1, \ldots, m$. In der Realität ist dem aber i.d.R. nicht so.

Bei linearen Ausgleichsproblem wird der Spezialfall berachtet, dass y eine *lineare* Funktion in den Parametern $x_1, \ldots, x_n$ ist:

$$y(t, x_1, \ldots, x_n) = \sum_{i=1}^{n} a_{ji}(t)x_i, \quad j = 1, \ldots, m \gg n,$$

wobei $a_{ij} : \mathbb{R} \to \mathbb{R}$ beliebige Funktionen sind. Mit $A = (a_{ij}) = (a_{ij}(i)) \in \mathbb{R}^{m\times n}$ und $b = (b_1, \ldots, b_m)^T$ erhält man das lineare Ausgleichsproblem (1.48).

Oft ist die Modellfunktion y aber *nichtlinear* in den Parametern $x_1, \ldots, x_n$. Dieser Fall ist Thema dieses Abschnitts. Zur Motivation betrachten wir wiederum ein Beispiel:

Beispiel 1.3.21 (Gedämpfte Schwingung) Schwingungen mit einem Freiheitsgrad werden im einfachsten Fall mathematisch durch eine Differentialgleichung zweiter Ordnung mit linearen Koeffizienten beschrieben

$$u'' + \frac{B}{M}u' + \frac{k}{M}u = 0,$$

wobei M die Masse, B die Dämpfungskonstante und k die Federsteifigkeit bezeichnet. Ohne genauer auf die mechanischen Hintergründe und die Lösungstheorie einzugehen halten wir fest, dass Lösungen dieser partiellen Differentialgleichung von der Form

$$u(t) = u_0 e^{-\delta t} \sin(\omega_0 t + \varphi_0)$$

sind. Wobei u_0 die Startamplitude, $\delta = -B/(2M)$ die Dämpfung, $\omega_0 = \sqrt{k/M}$ die die gedämpfte Eigenkreisfrequenz und φ_0 die Phasenverschiebung bezeichnet. Diese Lösung u hat somit vier Parameter. Als Modellfunktion y der gedämpften Schwingung verwenden wir daher den Ansatz

$$\begin{array}{rccccc} y(t, x_1, x_2, x_3, x_4) = & x_1 & & 2^{-x_2 t} & \sin(x_3 t + & x_4). \\ & \uparrow & & \uparrow & \uparrow & \uparrow \\ & \text{Startamplitude} & & \text{Dämpfung} & \text{Frequenz} & \text{Phasenverschiebung} \end{array}$$

Mit der **Methode der kleinsten Fehlerquadrate** ergibt sich:

$$\sum_{i=1}^{m} (y(t_i, x_1, x_2, x_3, x_4) - b_i)^2 = \|F(x_1, x_2, x_3, x_4)\|_2^2$$

mit

$$F(x) := y(x) - b, \quad y : \mathbb{R}^4 \to \mathbb{R}^m, b \in \mathbb{R}^m.$$

Man beachte y ist nichtlinear in x und daher auch F. Insbesondere ist F auch *nicht* quadratisch, also ist die Nullstelle des Gradienten nicht unbedingt ein (lokales) Extremum!

Nichtlineares Ausgleichsproblem: Allgemein setzen wir

$$F(x) := y(x) - b, \quad y : \mathbb{R}^n \to \mathbb{R}^m, b \in \mathbb{R}^m.$$

wobei F eine nichtlineare Funktion ist. Das nichtlineares Ausgleichsproblem lautet somit Für $F : \Omega \subset \mathbb{R}^n \to \mathbb{R}^m$, $F \in C^2(\Omega)$ suche $x \in \Omega$ mit

$$\hat{f}(x) := \|F(x)\|_2^2 \to \min . \tag{1.49}$$

Herleitung des Algorithmus: Die Idee ist es nun, deratige Probleme auf eine *Folge* linearer Ausgleichsprobleme zurück zuführen. Wie auch schon beim *Newton*-Verfahren ersetzt man nun $F(x)$ durch die Linearisierung $T(x)$

$$T(x) = F(x^{(k)}) + J_F(x^{(k)})(x - x^{(k)}).$$

Dadurch erhält man das lineare Ausgleichsproblem: bestimme $s^{(k)} \in \mathbb{R}^n$ so, dass

$$q_k(x) := \|T(x)\|_2^2 = \|F(x^{(k)}) + J_F(x^k)s^{(k)}\|_2^2 \to \min, \tag{1.50}$$

also $\|Ax-b\|_2$ mit $A = J_F(x^{(k)})$, $b = -F(x^{(k)})$, und setzen $x^{(k+1)} = x^{(k)}+s^{(k)}$. Aus Numerik 1 ist bekannt, dass man zu einem linearen Ausgleichsproblem die **Gauß'sche Normalengleichung** formulieren kann

$$2A^T Ax = 2A^T b,$$

welche der notwendigen Optimalitäsbedingung $\nabla f(x) = 0$ entspricht.

Algorithmus 1.3.3 (Gauß–Newton-Verfahren) *Wähle einen Startwert $x^{(0)}$ und eine Toleranz $\epsilon > 0$.*
Für $k = 0, 1, 2, \ldots$

1) *Berechne $F(x^{(k)}), J_F(x^{(k)})$*
2) *Bestimme $s^{(k)}$ durch Lösen der Gauß-Newton-Gleichung*

$$J_F(x^{(k)})^T J_F(x^{(k)})s^{(k)} = -J_F(x^{(k)})^T F(x^{(k)}).$$

3) *Setze $x^{(k+1)} = x^{(k)} + s^{(k)}$*

Bemerkung 1.3.22 *(a) Die Konvergenz ist i.A. linear, in speziellen Fällen quadratisch.*

(b) Man kann wiederum Dämpfungsstrategien anwenden.

(c) Eine bekannte Modifikation ist das Levenberg–Marquardt–Verfahren, das in vielen Statistik–Paketen verwendet wird (Numerik III).

2 INTERPOLATION

Häufig sind in der Praxis z.B. durch Marktanalysen, technische Messungen von einer Funktion nur einzelne Punkte bekannt, aber keine analytische Beschreibung der Funktion, um sie an beliebigen Stellen auswerten zu können. Könnte man die diskreten Daten durch eine (eventuell glatte) Kurve verbinden, so wäre es möglich, die unbekannte Funktion an den dazwischenliegenden Stellen zu schätzen. In anderen Fällen will man eine schwierig berechenbare Funktion näherungsweise durch eine einfachere darstellen. Eine Interpolationsfunktion kann diese Anforderung der Einfachheit erfüllen.

Interpolationsaufgabe: Eine gegebene Funktion $f : I \to \mathbb{R}$ sei geeignet zu approximieren unter der Vorgabe, dass f an diskreten (d.h. endlich vielen) Stützstellen die gegebenen Funktionswerte annehmen soll.

Die Interpolation ist somit eine Art der Approximation. Die Approximationsgüte hängt vom Ansatz ab. Um sie zu schätzen, werden Zusatzinformationen (Co-observations) über die Funktion f benötigt. Diese ergeben sich auch bei Unkenntnis von f häufig in natürlicher Weise: Beschränktheit, Stetigkeit oder Differenzierbarkeit lassen sich häufig voraussetzen.

Bei anderen Approximationsverfahren wie z. B. der Ausgleichungsrechnung wird nicht gefordert, dass die Daten exakt wiedergegeben werden; das unterscheidet diese Verfahren von der Interpolation.

Bemerkung 2.0.1 *i) Ist man am gesamten Verlauf von f interessiert, so sollte man eine Interpoliernde If konstruieren, die sich „möglichst wenig“ von f unterscheidet.*

ii) Diese **Interpolierende** *If sollte eine leicht berechenbare Funktion sein - hierfür eignen sich* **Polynome**, **trigonometrische Funktionen**, **Exponentialfunktionen** *sowie* **rationale Funktionen**.

2.1 DAS ALLGEMEINE INTERPOLATIONSPROBLEM

Manchmal möchte man nicht nur Daten (also Funktionswerte) interpolieren, sondern z.B. auch Steigungen, Krümmungen oder Richtungsableitungen. Dazu ist folgende allgemeine Problemformulierung hilfreich.

Definition 2.1.1 (Lineares Funktional) *Sei $\mathcal{F}$ ein linearer Raum (z.B. ein Funktionenraum $(C[a,b]$, $C^\infty(\Omega)$, $L_2(\Omega)$, $\ldots)$. Eine Abbildung $\mu : \mathcal{F} \to \mathbb{R}$ heißt* ***lineares Funktional****, falls*

$$\mu(\alpha_1 f_1 + \alpha_2 f_2) = \alpha_1 \mu(f_1) + \alpha_2 \mu(f_2), \quad \alpha_1, \alpha_2 \in \mathbb{R},\ f_1, f_2 \in \mathcal{F}, \tag{2.1}$$

d.h. falls μ reellwertig und linear ist.

Beispiel 2.1.2 Folgende Funktionen sind lineare Funktionale:

(a) $\mathcal{F} = C[a,b],\ x_0 \in [a,b],\ \mu(f) := f(x_0)$, Funktionswerte, Punktauswertungen.

(b) $\mathcal{F} = C^1[a,b],\ x_0 \in [a,b],\ \mu(f) := f'(x_0)$, Steigungen, Ableitungen.

(c) $\mathcal{F} = R[a,b]$,

$$\mu(f) := \int_a^b f(x)\,dx,$$

Integrale.

(d) Mittelwerte, Mediane und andere statistische Größen.

Dann lautet das **allgemeine Interpolationsproblem:** Sei $\mathcal{F}$ ein Funktionenraum und G_n ein $(n+1)$-dimensionaler Teilraum von $\mathcal{F}$. Weiter seien lineare Funktionale $\mu_0, \ldots, \mu_n$ auf G_n gegeben.

$$\left\{ \begin{array}{ll} \text{Zu } f \in \mathcal{F} & \text{finde } g_n \in G_n \text{ mit} \\ & \mu_j(g_n) = \mu_j(f), \quad j = 0, \ldots, n. \end{array} \right. \tag{2.2}$$

Man kann nun die Frage der Existenz und Eindeutigkeit beweisen:

Lemma 2.1.3 (Existenz und Eindeutigkeit der Lösung der allgemeinen Interpolationsaufgabe) *Die allgemeine Interpolationsaufgabe (2.2) hat genau dann für jedes $f \in \mathcal{F}$ eine eindeutige Lösung g_n, wenn*

$$\det[(\mu_i(\varphi_j))_{i,j=0}^n] \neq 0$$

für eine (und damit jede) Basis $\{\varphi_0, \ldots, \varphi_n\}$ von G_n gilt.

Beweis. Sei $g_n \in G_n$, dann existieren eindeutig bestimmte Koeffizienten $\{\alpha_0, \ldots, \alpha_n\}$ mit $g_n = \sum_{j=0}^n \alpha_j \varphi_j$, also bedeutet (2.2)

$$\mu_i(g_n) \quad = \quad \sum_{j=0}^n \alpha_j \mu_i(\varphi_j) = (G\alpha)_i \overset{(2.2)}{=} \mu_i(f) = b_i$$

mit $G = (\mu_i(\varphi_j))_{i,j=0}^n$, $\alpha = (\alpha_j)_{j=0}^n$, $b = (b_j)_{j=0}^n$, d.h. $G\alpha = b$. □

Bemerkung 2.1.4 *Je nach Wahl von $g_n \in G_n$ erhält man unterschiedliche Instanzen der Interpolation:*

- *Ist $G_n = \mathbb{P}_n$ redet man von **Polynominterpolation**.*
- *$G_{2n} := \left\{ F(x) := \frac{P(x)}{Q(x)}, P, Q \in \mathbb{P}_n, Q \neq 0 \right\}$ ergibt die **rationale Interpolation**.*
- *Ist $G_n = \mathcal{S}^k(\mathcal{T})$ so spricht man von **Spline-Interpolation** (Vgl. Kapitel 4).*

2.2 POLYNOMINTERPOLATION

Nach dem aus der Analysis bekannten Approximationssatz von Weierstraßist es möglich, jede stetige Funktion beliebig gut durch Polynome zu approximieren. In diesem Abschnitt sei $G_n = \mathbb{P}_n$, d.h. der Vektorraum der Polynome vom Grad kleiner gleich n.

Wenn man nun in der allgemeinen Interpolationsaufgabe (2.2) nur Funktionswerte f_i interpoliert, spricht man von ***Lagrange*-Interpolation**. Werden Funktionswerte und Ableitungen interpoliert, spricht man von ***Hermite*-Interpolation**. Beide Interpolationsaufgaben werden im Folgenden behandelt.

2.2.1 Lagrange-Interpolation

Gegeben seien $(n+1)$ diskrete, paarweise verschiedene **Stützstellen** (auch **Knoten** genannt) $x_0, \ldots, x_n$ und dazugehörige beliebige **Stützwerte** $f_0, \ldots, f_n$.

Gesucht ist nun ein Polynom $P \in \mathbb{P}_n$ vom Grad grad $p \leq n$, d.h.

$$P(x) = a_n x^n + \ldots + a_1 x + a_0, \qquad \text{mit } a_\nu \in \mathbb{R}, \nu = 0, \ldots, n,$$

welches die **Interpolationsbedingungen**

$$P(x_i) = f_i, \qquad i = 0, \ldots, n \tag{2.3}$$

erfüllt.

Sei $\varphi_j \equiv x^j, j = 0, \ldots, n$ die **monomiale Basis** des $\mathbb{P}_n$. Für (2.3) erhalten wir $\mu_i(f) := f(x_i)$ also $\mu_i(\varphi_j) = x_i^j$. Also ist die Matrix $(\mu_i(\varphi_j))_{i,j=0}^n$ die Vandermonde-Matrix, die bekanntermaßen invertierbar ist, falls die x_i paarweise disjunkt sind.

Um eines solches Polynoms $P(x)$ auch explizit zu konstruieren, werden wir zunächst die sogenannte *Lagrange*-Darstellung von Polynomen einführen. Mit dieser Darstellung kann man besonders schön zeigen, dass man immer ein Polynom konstruieren kann, welches die Interpolationsaufgabe erfüllt.

2.2.1.1 *Lagrange*-Darstellung

Zur Konstruktion der *Lagrange*-Darstellung von Polynomen benötigen wir die folgende Definition:

Definition 2.2.1 (*Lagrange*-Polynome) *Die Polynome*

$$L_i^{(n)}(x) := \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{(x - x_j)}{(x_i - x_j)} \in \mathbb{P}_n, \quad i = 0, 1, \ldots, n. \tag{2.4}$$

zu den $(n+1)$ diskreten, paarweise verschiedenen Knoten $x_0, \ldots, x_n$ werden **Lagrange-*Basispolynome*** *genannt.*

Bemerkung 2.2.2 (Basis-Darstellung) *Als endlich dimensionalen Vektorraum kann man die Elemente des Vektorraums $\mathbb{P}_n$ der Polynome vom Grad kleiner oder gleich n, d.h., die Polynome, eindeutig als Linearkombination endlicher, linear unabhängiger Teilmengen—einer Basis–darstellen. Wie man sich leicht überlegt, ist die Menge der* Lagrange-*Polynome $\{L_i^{(n)}, i = 0, \ldots, n\}$ eine Basis des $\mathbb{P}_n$ (Übungsaufgabe).*

Bemerkung 2.2.3 (Eigenschaft der *Lagrange*-Polynome) *Per Konstruktion erfüllen die* Lagrange-*Polynome die Knotenbedingung*

$$L_i^{(n)}(x_k) = \delta_{ik} = \begin{cases} 1 & \text{für } i = k, \\ 0 & \text{für } i \neq k. \end{cases} \tag{2.5}$$

Hieraus folgt, das $L_i^{(n)}, i = 0, \ldots, n$ linear unabhängig sind.

Definition 2.2.4 (Lagrange-Darstellung) *Das Polynom*

$$P(x) := \sum_{i=0}^{n} f_i L_i^{(n)}(x) \tag{2.6}$$

wird ***Lagrange-Darstellung*** *des Interpolationspolynoms zu den Stützstellen $(x_0, f_0), \ldots, (x_n, f_n)$ genannt.*

Wie angekündigt, liefert die *Lagrange*-Darstellung einen konstruktiven Beweis für die Existenz und Eindeutigkeit eines Polynoms $P(x)$, das die Interpolationsbedingung (2.3) erfüllt.

Satz 2.2.5 (Existenz und Eindeutigkeit der *Lagrange*-Interpolation) *Zu beliebigen $(n+1)$ Paaren (x_i, f_i) , $i = 0, \ldots, n$, mit paarweise verschiedenen Stützstellen $x_0, \ldots, x_n$ existiert genau ein Interpolationspolynom $P \in \mathbb{P}_n$, das (2.3) erfüllt.*

Beweis. **(Existenz:)** Die Existenz des Interpolationspolynoms $P(x)$ zeigen wir auf konstruktive Art. Zu diesem Zweck betrachten wir zu den gegebenen Stützstellen die $(n+1)$ ***Lagrange*-Polynome** $L_i^{(n)}, i = 0, \ldots, n$, aus (2.4). Diese Polynome sind vom echten Grad n und besitzen offensichtlich die Eigenschaft (2.5). Demzufolge besitzt das Polynom

$$P(x) := \sum_{i=0}^{n} f_i L_i^{(n)}(x)$$

wegen (2.5) die geforderten Interpolationseigenschaften:

$$P(x_k) = \sum_{i=0}^{n} f_i L_i^{(n)}(x_k) = \sum_{i=0}^{n} f_i \delta_{ik} = f_k, \quad k = 0, 1, \ldots, n.$$

Ferner ist als Linearkombination von Polynomen vom Grad n der Grad von P kleiner oder gleich n.

(Eindeutigkeit:) Die Eindeutigkeit des Interpolationspolynoms ergibt sich wie folgt: Es seien P und Q zwei Polynome jeweils vom Grad höchstens gleich n, $P, Q \in \mathbb{P}_n$ mit

$$P(x_k) = Q(x_k) = f_k \quad (k = 0, 1, \ldots, n). \tag{2.7}$$

Aus dieser Eigenschaft (2.7) folgt, dass $D(x) := P(x) - Q(x)$ ein Polynom vom Grad kleiner oder gleich n definiert mit den $(n+1)$ paarweise verschiedenen Nullstellen $x_0, \ldots, x_n$. Nach dem Fundamentalsatz der Algebra muss nun aber $D(x) \equiv 0$ gelten, also $P(x) = Q(x)$ sein. □

Bemerkung 2.2.6 (Vor- und Nachteile der *Lagrange*-Darstellung) *Die* Lagrange-*Darstellung (2.6) ist für praktische Zwecke meist zu rechenaufwendig, sie eignet sich jedoch hervorragend für theoretische Fragestellungen. Insbesondere, müssen die Lagrange-Polynome bei Hinzunahme einer weiteren Stützstelle (x_{n+1}, f_{n+1}) komplett neu berechnet werden.*

Im Laufe dieses Abschnitts werden wir eine weitere Basis Darstellung kennenlernen, welche sich besser zur numerischen Berechnung eignet, als die *Lagrange*-Darstellung, dies ist die so genannte *Newton*-Darstelleung. Zunächst aber zur kanonischen Basis Darstellung.

2.2.1.2 Monomiale Basis Darstellung

Eine alternative Basis zur Darstellung des Interpolationspolynoms mit Lagrange-Polynomen bildet die sogenannte **monomiale Basis** $\{1, \ldots, x^n\}$ von $\mathbb{P}_n$, d.h. wir schreiben P in folgender Koeffizientendarstellung

$$P(x) = a_0 + a_1 x + \ldots + a_n x^n.$$

Bemerkung 2.2.7 (*Vandermonde*-Matrix) *Die Interpolationsbedingungen $P(x_i) = f_i$, $i = 0, \ldots, n$ lassen sich wie bereits bemerkt als folgendes lineares Gleichungssystem auffassen:*

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^n \\ 1 & x_1 & x_1^2 & \ldots & x_1^n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_n & x_n^2 & \ldots & x_n^n \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} f_0 \\ f_1 \\ \vdots \\ f_n \end{pmatrix}.$$

In Numerik 1 haben wir diese Vandermonde*-Matrix schon kennengelernt und an dortiger Stelle auch eine zum* Gauß*-Algorithmus alternative Möglichkeit angegeben um das dazugehörige lineare Gleichungssystem zu lösen (Aufwand für dieses spezielle Verfahren $5n^2/2 + \mathcal{O}(n)$). Das Problem der schlechten Kondition hatten wir auch angesprochen.*

Bemerkung 2.2.8 (Nachteil monomiale Basis Darstellung) *Die Darstellung in monomialer Basis ist numerisch instabil und sollte daher für große n im Allgemeinen nicht verwendet werden.*

2.2.2 Hermite-Interpolation

Um die *Hermite*-Interpolation einzuführen, bei der neben den Funktionswerten $f(x_i)$ auch die Ableitungen gegeben sind, benötigen wir noch einige Notationen. Zunächst sei angemerkt, dass generell Ableitungen verschiedener Ordnung an unterschiedlichen Knoten gegeben sein können.

Notation: Wir definieren die Folge von Stützstellen $\triangle := \{x_j\}_{j=0,\dots,n}$ mit

$$a = x_0 \leq x_1 \leq \cdots \leq x_n = b,$$

wobei Stützstellen auch mehrfach auftreten können. Sind an einer Stelle x_i einerseits der Funktionswert $f(x_i)$ und andererseits die Ableitungen $f'(x_i), \dots, f^{(k)}(x_i)$ gegeben, so soll x_i in obiger Folge $(k+1)$-mal auftreten.

Gleiche Knoten nummerieren wir hierbei mit der Vielfachheit

$$d_i := \max\{j \in \mathbb{N} \mid x_i = x_{i-j}\}$$

von links nach rechts durch; z.B.

x_i	$x_0 = x_1$	$<$	$x_2 = x_3 = x_4$	$<$	x_5	$<$	x_6
d_i	0	1	0	1	2	0	0

Führen wir nun mit diesen Abkürzungen nachfolgende lineare Abbildung

$$\mu_i : C^n[a,b] \to \mathbb{R}, \quad \mu_i(f) := f^{(d_i)}(x_i), \quad i = 0, \dots, n$$

ein, so lautet die **Aufgabe der *Hermite*-Interpolation:** Gegeben μ_i ($i = 0, \dots, n$). Finde $P \in \mathbb{P}_n$ mit

$$\mu_i(P) = \mu_i(f), \quad i = 0, \dots, n. \tag{2.8}$$

Die Lösung $P = P(f|x_0, \dots, x_n) \in \mathbb{P}_n$ von (2.8) heißt ***Hermite*-Interpolierende**.

Satz 2.2.9 (Existenz und Eindeutigkeit der *Hermite*-Interpolation) *Zu jeder Funktion $f \in C^n[a,b]$ und jeder monotonen Folge*

$$a = x_0 \leq x_1 \leq \ldots \leq x_n = b$$

von (i.Allg. nicht paarweise verschiedenen) Knoten gibt es genau ein Polynom $P \in \mathbb{P}_n$, sodass gilt:

$$\mu_i(P) = \mu_i(f), \quad i = 0, \dots, n.$$

Beweis. Die Abbildung

$$\mu : \mathbb{P}_n \to \mathbb{R}^{n+1}, \; P \mapsto (\mu_0(P), \dots, \mu_n(P))$$

ist offensichtlich eine lineare Abbildung zwischen den $(n+1)$-dimensionalen reellen Vektorräumen $\mathbb{P}_n$ und $\mathbb{R}^{n+1}$, sodass aus der Injektivität der Abbildung bereits die Surjektivität folgen würde

und damit der Satz bewiesen wäre. Somit reicht es die Injektivität der linearen Abbildung zu zeigen.

Da $\mu(P) = 0$ gilt, folgt, dass p mindestens $(n+1)$-Nullstellen inklusive Vielfachheiten besitzt, somit aber das Nullpolynom ist. Da ferner $dim\, \mathbb{P}_n = dim\, \mathbb{R}^{n+1} = n+1$, folgt daraus auch wieder die Existenz. □

Bemerkung 2.2.10 (Eigenschaften der *Hermite*-Interpolation) *Stimmen alle Knoten überein, d.h. $x_0 = x_1 = \cdots = x_n$, dann entspricht das Interpolationspolynom der abgebrochenen Taylor-Reihe um $x = x_0$:*

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

Die *Hermite*-Interpolation und insbesondere deren effiziente Berechnung sind Thema des nun folgenden Abschnitts. Die dort vorgestellten Schemata zur effizienten numerischen Auswertung sind, wie wir an einem Beispiel sehen werden, allerdings natürlich auch im Fall der *Lagrange*-Interpolation anwendbar.

2.2.3 Auswertung von Polynomen

In diesem Abschnitt werden wir uns unter anderem mit der effizienten Auswertung des so genannten Interpolationspolynoms beschäftigen:

Definition 2.2.11 (Interpolationspolynom) *Das nach Satz 2.2.5 eindeutig bestimmte Polynom $P \in \mathbb{P}_n$ heißt **Interpolationspolynom** von f zu den paarweise verschiedenen Stützstellen $x_0, \ldots, x_n$ und wird mit*

$$P = P(f|x_0, \ldots, x_n)$$

bezeichnet.

Zur Berechnung kommen je nach Fragestellung zwei unterschiedliche Ansätze zum Einsatz:

- Schema von *Aitken*[1] und *Neville*[2],
- Schema der dividierten Differenzen.

Ist man nur an der Auswertung des Interpolationspolynoms P an einer Stelle x (oder ganz wenigen) interessiert und will das Interpolationspolynom selber nicht explizit ausrechnen so verwendet man das Schema von *Aitken* und *Neville*.

Will man hingegen das Interpolationspolynom an mehreren Stellen auswerten, so wendet man das Schema der dividierten Differenzen an. Hierbei berechnet man die Koeffezienten $a_n = [x_0, \ldots, x_n]f$ des so genannten Newtonschen-Interpolationspolynoms (die dividierten Differenzen) und wendet ein dem Horner-Schema ähnliches Schema an.

Das *Newton*sche-Interpolationspolynom ist hierbei eine Darstellung des Interpolationspolynoms bezüglich der so genannten *Newton*-Basis. Aus dem Blickwinkel der Darstellung des Interpolationspolynoms bzgl. der *Newton*-Basis entspricht das Schema von *Aitken* und *Neville* der Berechnung des Wertes des Interpolationspolynoms an einer gesuchten Stelle ohne die Koeffizienten a_n explizit zu berechnen.

[1] Aitken, Alexander Craig (1895-1967)
[2] Neville, Eric Harold (1889-1961)

2.2.3.1 Schema von *Aitken* und *Neville*

Hier betrachen wir den Fall, dass man nur an der Auswertung des Interpolationspolynoms P an einer Stelle x interessiert ist. Dazu muss man nicht erst P bestimmen, sondern kann $P(x)$ durch rekursive Berechnung effektiver (d.h vor allem effektiver bezüglich des Aufwands) bestimmen. Motiviert wird dies im Folgenden durch das Lemma von *Aitken*.

Lemma 2.2.12 (von Aitken) *Für das Interpolationspolynom $P = P(f|x_0, \ldots, x_n)$ gilt die Rekursionsformel*

$$P(f|x_0, \ldots, x_n)(x) = \frac{(x_0 - x)P(f|x_1, \ldots, x_n)(x) - (x_n - x)P(f|x_0, \ldots, x_{n-1})(x)}{x_0 - x_n}. \quad (2.9)$$

Hierbei gilt also insbesondere $P(f|x_k) = f(x_k)$.

Beweis. Sei $\phi(x)$ definiert als der Term auf der rechten Seite von (2.9). Dann ist $\phi \in \mathbb{P}_n$ und es gilt:

$$\phi(x_i) = \frac{(x_0 - x_i)f(x_i) - (x_n - x_i)f(x_i)}{x_0 - x_n} = f(x_i), \quad i = 1, \ldots, n-1.$$

Ebenso leicht folgt $\phi(x_0) = f(x_0)$ sowie $\phi(x_n) = f(x_n)$ und daher obige Behauptung. □

Herleitung des Algorithmus: Wir definieren $f_i := f(x_i)$ für $i = 0, \ldots, n$. Man beachte hierbei

$$P(f|x_i) = f_i, \quad i = 0, \ldots, n.$$

Für festes x vereinfachen wir die Notation weiterhin durch

$$P_{i,k} := P(f|x_{i-k}, \ldots, x_i)(x), \quad i \geq k.$$

Die Rekursion (2.9) schreibt sich nun

$$P_{n,n} = \frac{(x_0 - x)P_{n,n-1} - (x_n - x)P_{n-1,n-1}}{x_0 - x_n},$$

oder allgemeiner für $P_{i,k}$, $i \geq k$:

$$\begin{aligned} P_{i,k} &= \frac{\overbrace{(x_{i-k} - x)}^{=(x_{i-k}-x_i+x_i-x)} P_{i,k-1} - (x_i - x)P_{i-1,k-1}}{x_{i-k} - x_i} \\ &= P_{i,k-1} + \frac{x_i - x}{x_{i-k} - x_i}(P_{i,k-1} - P_{i-1,k-1}). \end{aligned}$$

Daraus gewinnen wir den folgenden Algorithmus:

Algorithmus 2.2.1 (*Aitken-Neville*-Verfahren) *Gegeben seien Stützstellen $x_0, x_1, \ldots, x_n$ und Stützwerte $f_0, f_1, \ldots, f_n$.*

1) *Setze $P_{i,0} = f_i$, für i=0,..., n.*

2) *Berechne $P_{i,k} = P_{i,k-1} + \frac{x - x_i}{x_i - x_{i-k}}(P_{i,k-1} - P_{i-1,k-1})$, $j \geq k$.*

Zur praktischen Berechnung eignet sich das **Schema von *Neville***: Nach diesem lässt sich $P_{n,n} = P(f|x_0, \ldots, x_n)(x)$, d.h. das Interpolationspolynom an einer festen Stelle x, ausgehend von den Daten $(f_0 \ldots, f_n)$ wie folgt berechnen:

$$
\begin{array}{ccccccccccc}
f_0 & = & P_{0,0} & & & & & & & & \\
 & & & \searrow & & & & & & & \\
f_1 & = & P_{1,0} & \to & P_{1,1} & & & & & & \\
\vdots & & \vdots & & \vdots & & & & & & \\
\vdots & & \vdots & & & & & & & & \\
 & & & & & \searrow & & & & & \\
\vdots & & \vdots & & \vdots & & P_{n-2,n-2} & & & & \\
 & & & & & \searrow & & \searrow & & & \\
f_{n-1} & = & P_{n-1,0} & \to & \cdots & \to & P_{n-1,n-2} & \to & P_{n-1,n-1} & & \\
 & & & \searrow & & \searrow & & \searrow & & \searrow & \\
f_n & = & P_{n,0} & \to & P_{n,1} \cdots & \to & P_{n,n-2} & \to & P_{n,n-1} & \to & P_{n,n}
\end{array}
$$

MATLAB-Funktion: AitkenNeville.m

```
function value = AitkenNeville(x,fx,x0)
% evaluate the Interpolation polynomial given
% by (x,fx) at the point x0
for k = 2:length(fx)
  for j = length(fx):-1:k
    fx(j) = fx(j) + (x0-x(j))/(x(j)-x(j-k+1)) * (fx(j)-fx(j-1));
  end
end
value = fx(end);
```

MATLAB-Beispiel:

Testen wir das *Aitken-Neville*-Verfahren anhand zweier Beispiele. Zum einen werten wir das Interpolationspolynom zu $f(x) = x^2$ und 3 Stützstellen an der Stelle 2 aus.

```
>> x = linspace(0,1,5);
>> AitkenNeville(x,x.^2,2)
ans =
     4
```

Zum anderen werten wir das Interpolationspolynom zu $f(x) = \sin(x)$ und 5 äquidistanten Stützstellen in $[0, 1]$ an der Stelle $\pi/3$ aus. Man beachte, dass $\sin(x) = \sqrt{3}/2$ gilt.

```
>> x = linspace(0,1,5);
>> sqrt(3)/2 - AitkenNeville(x,sin(x),pi/3)
ans =
    4.387286117690792e-005
```

Bemerkung 2.2.13 *Im Gegensatz zur Lagrange-Darstellung lässt sich bei der Berechnung des Interpolationspolynoms an einer festen Stelle mit dem Schema von Aitken-Neville zu $(n+1)$-Paaren ohne Probleme eine weiteres Paar (x_{n+1}, f_{n+1}) hinzu nehmen. Und dies somit zu einer Berechnung des Interpolationspolynoms an einer festen Stelle zu $(n+2)$ Paaren ausweiten.*

2.2.3.2 Newton-Darstellung und dividierte Differenzen

Wir hatten bereits zwei Basis-Darstellungen für $\mathbb{P}_n$ kennengelernt. Allerdings eigenen sich sowohl die *Lagrange*-Darstellung als auch die monomiale Basis Darstellung nicht so gut zur numerischen Berechnung des kompletten Polynoms (also nicht nur der Auswertung an wenigen Stellen). Vor diesem Hintergrund führen wir die *Newton*schen Basispolynome ein, eine Darstellung bzgl. dieser ist für numerische Zwecke besser geeignet.

Das Ziel ist eine Aufdatierungsstrategie bezüglich des Polynomgrades zur Berechnung des kompletten Polynoms. Mit einer Aufdatierungsstrategie kann man dann auch ohne Probleme ein weiteres Paar (x_{n+1}, f_{n+1}) hinzunehmen.

Motivation: Zu gegebenen $(n+1)$ Paaren ist wollen wir das Interpolationspolynom $P(f|x_0,\dots,x_n) \in \mathbb{P}_n$ somit als eine additive Zerlegung

$$\underbrace{P(f|x_0,\dots,x_n)(x)}_{\in\mathbb{P}_n} = \underbrace{P(f|x_0,\dots,x_{n-1})(x)}_{\in\mathbb{P}_{n-1}} + \underbrace{Q_n(x)}_{\in\mathbb{P}_n}$$

darstellen, mit einem Polynom Q_n vom Grad n, welches von den Stützstellen x_i und einem unbekannten Koeffizienten abhängt. Da für

$$Q_n(x_i) = P(f|x_0,\dots,x_n)(x_i) - P(f|x_0,\dots,x_{n-1})(x_i) = 0$$

gilt, muss

$$Q_n(x) = a_n(x-x_0)\cdots(x-x_{n-1})$$

gelten. Für x_n setzt man ein und löst nach a_n auf

$$a_n = \frac{f(x_n) - P(f|x_0,\dots,x_{n-1})(x_n)}{(x_n-x_0)\cdots(x_n-x_{n-1})}$$

Offenbar ist a_n der führende Koeffizient von $P(f|x_0,\dots,x_n)$, d.h. der Koeffizient vor x^n.

Wir definieren nun die bereits erwähnte Newton-Basis:

Definition 2.2.14 (*Newton*-Basis) *Es seien $x_0,\dots x_{n-1} \in \mathbb{R}$ und*

$$\omega_0 := 1, \quad \omega_i(x) := \prod_{j=0}^{i-1}(x-x_j), \quad \omega_i \in \mathbb{P}_i.$$

Wir bezeichnen $\{\omega_0,\dots,\omega_n\}$ als **Newton-*Basis*** *des Polynomraums $\mathbb{P}_n$ mit Basiselementen ω_i.*

Bemerkung 2.2.15 *Man beachte, dass bei der Definition der Newton-Basis weder eine Ordnung der Punkte x_k noch „paarweise verschieden“ vorgeschrieben wurde. Je nach Nummerierung, erhält man somit eine andere Newton-Basis. Dies ist bei der Stabilität der folgenden Verfahren zu berücksichtigen.*

Um nun das Verfahren der dividierten Differenzen herzuleiten, mit dem sich sowohl die *Lagrange*- als auch die *Hermite*-Interpolationsaufgabe effizient lösen lässt, verwenden wir die Darstellung des Interpolationspolynoms in der *Newton*-Basis, d.h. das ***Newton*-Interpolationspolynom**

$$\begin{aligned} P(x) &= c_0 + c_1(x-x_0) + c_2(x-x_0)(x-x_1) + \cdots + c_n \prod_{j=0}^{n-1}(x-x_j) \\ &= \sum_{i=0}^{n} a_i \omega_i(x), \end{aligned}$$

wobei sich die unbekannten Koeffizienten $c_0, \ldots, c_n$ prinzipiell aus den Interpolationsbedingungen

$$\begin{array}{rcll} P(x_0) &=& c_0 & = f(x_0) \\ P(x_1) &=& c_0 + c_1(x-x_0) & = f(x_1) \\ P(x_2) &=& c_0 + c_1(x_2-x_0) + c_2(x_2-x_1)(x_2-x_0) & = f(x_2) \\ \vdots & & \vdots & \end{array}$$

sukzessive berechnen lassen (dieses LGS hat Linksdreiecksgestalt!):

$$\begin{aligned} P(x_0) = c_0 &\quad\Longrightarrow\quad c_0 = f(x_0), \\ P(x_1) = c_0 + c_1(x_1 - x_0) &\quad\Longrightarrow\quad c_1 = \frac{f(x_1) - f(x_0)}{x_1 - x_0} \end{aligned}$$

Die Koeffizienten $c_k, k = 0, \ldots, n$ nennt man **dividierte Differenzen**.

Definition 2.2.16 (n-te dividierte Differenz) *Der führende Koeffizient a_n des Interpolationspolynoms*

$$P(f|x_0, \ldots, x_n)(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$$

*von f zu den Knoten $x_0 \le x_1 \le \cdots \le x_n$ heißt **n-te dividierte Differenz** von f an $x_0, \ldots, x_n$ und wird mit*

$$[x_0, \ldots, x_n]f := a_n$$

bezeichnet.

Bemerkung 2.2.17 *Beim Vergleich des* Newton-*Interpolationspolynoms mit dem Interpolationspolynom bzgl. der monomialen Basis*

$$P(x) = \sum_{j=0}^{n} c_i \prod_{i=0}^{j-1}(x - x_i) = \sum_{j=0}^{n} a_j x^j$$

sieht man durch Ausmultiplizieren, dass für die Koeffizienten der höchsten $x-$Potenz $a_n = c_n$ gilt. Damit folgt $a_n = c_n = [x_0, \ldots, x_n]f$

Satz 2.2.18 (*Newton*'sche Interpolationsformel) *Für jede Funktion $f \in C^n(\mathbb{R})$ und Knoten $x_0 \le \cdots \le x_n \in \mathbb{R}$ ist*

$$P(x) = \sum_{i=0}^{n} [x_0, \ldots, x_i]f \cdot \omega_i(x)$$

das Interpolationspolynom $P(f|x_0, \ldots, x_n)$ von f an $x_0, \ldots, x_n$ also $P(f|x_0, \ldots, x_n) = P(x)$. Gilt darüber hinaus $f \in C^{n+1}(\mathbb{R})$, so folgt:

$$f(x) = P(x) + [x_0, \ldots, x_n, x]f \cdot \omega_{n+1}(x)\,. \tag{2.10}$$

Beweis. Wir zeigen die erste Behauptung durch Induktion nach $n \in \mathbb{N}$. Für $n = 0$ ist die Aussage trivialerweise erfüllt. Sei also im Folgenden $n > 0$ und

$$P_{n-1} := P(f|x_0, \ldots, x_{n-1}) = \sum_{i=0}^{n-1} [x_0, \ldots, x_i]f \cdot \omega_i$$

das Interpolationspolynom von f an den Stützstellen $x_0, \ldots, x_{n-1}$. Damit erhalten wir für $P_n = P(f|x_0, \ldots, x_n)$, aufgrund der Definition der dividierten Differenz bzw. von P_n, dass

$$\begin{aligned} P_n(x) &= [x_0, \ldots, x_n]f \cdot x^n + a_{n-1}x^{n-1} + \ldots + a_0 \\ &= [x_0, \ldots, x_n]f \cdot \omega_n(x) + Q_{n-1}(x) \end{aligned}$$

mit einem Polynom $Q_{n-1} \in \mathbb{P}_{n-1}$ gilt. Nun erfüllt aber wegen $\omega_n(x_i) = 0, i = 0, \ldots, n-1$

$$Q_{n-1} = P_n - [x_0, \ldots, x_n]f \cdot \omega_n$$

offensichtlich die Interpolationsaufgabe für $x_0, \ldots, x_{n-1}$, sodass wir erhalten:

$$Q_{n-1} = P_{n-1} = \sum_{i=0}^{n-1} [x_0, \ldots, x_i]f \cdot \omega_i.$$

Dies beweist aber gerade die Aussage des Satzes. Insbesondere folgt nun, dass

$$P_n + [x_0, \ldots, x_n, x]f \cdot \omega_{n+1}$$

die Funktion f an den Knoten $x_0, \ldots, x_n$ und x interpoliert und damit (2.10). □

Aus den Eigenschaften der *Hermite*-Interpolation lassen sich sofort folgende Aussagen über die dividierten Differenzen zu f ableiten:

Lemma 2.2.19 (Eigenschaften der dividierten Differenzen) *i)* ***(Rekursive Berechnung der dividierten Differenzen)*** *Für $x_i \neq x_k$ gilt die Rekursionsformel*

$$[x_0, \ldots, x_n]f = \frac{[x_0, \ldots, \widehat{x}_i, \ldots, x_n]f - [x_0, \ldots, \widehat{x}_k, \ldots, x_n]f}{x_k - x_i},$$

wobei $\widehat{\ }$ anzeigt, dass die entsprechende Stützstelle weggelassen wird ('seinen Hut nehmen muss')

$ii)$ ***(Darstellung über abgebrochene Taylor-Reihe)*** *Für zusammenfallende Knoten $x_0 = \ldots = x_n$ gilt*

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

Bemerkung 2.2.20 *Aussage $i)$ erklärt die Bezeichnung dividierte Differenzen.*

Beweis. Für das *Hermite*-Interpolationspolynom gilt mit $x_i \neq x_k$:

$$P(f|x_0, \ldots, x_n) = \frac{(x_i - x)\overbrace{P(f|x_0, \ldots, \widehat{x}_k, \ldots, x_n)}^{[x_0, \ldots, \widehat{x}_i, \ldots, x_n]f\, x^{n-1} + \mathbb{P}_{n-2}} - (x_k - x)P(f|x_0, \ldots, \widehat{x}_k, \ldots, x_n)}{x_i - x_k}, \tag{2.11}$$

was sich durch Überprüfen der Interpolationseigenschaft zeigen lässt mittels Einsetzen der Definitionen. Aus der Eindeutigkeit des führenden Koeffizienten folgt aus (2.11) unmittelbar Behauptung $i)$. Stimmen dagegen alle Knoten $x_0, \ldots, x_n$ überein, so ist das Interpolationspolynom

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

wie man durch Einsetzen in μ_i (vgl. (2.8) oben) leicht einsieht. Daraus folgt, dass der führende Koeffizient $f^{(n)}(x_0)/n!$ ist. Somit gilt auch $ii)$. □

Die Auswertung der Rekursionsformel (2.11) erfolgt am zweckmäßigsten im **Schema der dividierten Differenzen** unter Verwendung der Startwerte $[x_i]f = f(x_i)$ für paarweise verschiedene Knoten.

$$
\begin{array}{lllll}
x_0 & [x_0]f & & & \\
x_1 & [x_1]f & [x_0,x_1]f & & \\
x_2 & [x_2]f & [x_1,x_2]f & [x_0,x_1,x_2]f & \\
x_3 & [x_3]f & [x_2,x_3]f & [x_1,x_2,x_3]f & [x_0,x_1,x_2,x_3]f \\
x_4 & [x_4]f & [x_3,x_4]f & [x_2,x_3,x_4]f & [x_1,x_2,x_3,x_4]f \quad [x_0,\ldots,x_4]f
\end{array}
$$

Die gesuchten Koeffizienten a_k des *Newton*-Interpolationspolynoms findet man im obigen Schema der dividierten Differenzen in der oberen Diagonalen.

Nachfolgend werden wir das Schema der dividierten Differenzen anhand zweier Beispiele illustrieren. Zum einen für die klassische *Lagrange*-Interpolation bei der Knoten und Funktionswerte gegeben sind und zum anderen für die *Hermite*-Interpolation.

Beispiel 2.2.21 (*Lagrange*-Interpolation via dem Schema der dividierten Differenzen) Wenn wir das Schema auf gegebene Daten $(x_i, f_i), i = 0,\ldots,3$ anwenden, so erhalten wir

$$
\begin{array}{lcl}
x_0 = 0 & : & f_0 = 1 \\
 & & \searrow \\
x_1 = 3/2 & : & f_1 = 2 \quad \to \quad 2/3 \\
 & & \searrow \qquad\qquad \searrow \\
x_2 = 5/2 & : & f_2 = 2 \quad \to \quad 0 \quad \to \quad -4/15 \\
 & & \searrow \qquad\qquad \searrow \qquad\qquad \searrow \\
x_3 = 9/2 & : & f_3 = 1 \quad \to \quad -1/2 \quad \to \quad -1/6 \quad \to \quad 1/45
\end{array}
$$

und das *Newtonsche* Interpolationspolynom lautet demnach

$$P(x) = 1 + \frac{2}{3}(x-0) - \frac{4}{15}(x-0)(x-\frac{3}{2}) + \frac{1}{45}(x-0)(x-\frac{3}{2})(x-\frac{5}{2})$$

MATLAB-Funktionen: NewtonInterpolation.m und EvalNewtonPoly.m

```
function fx = NewtonInterpolation(x,fx)
for k = 2:length(fx)
  for j = length(fx):-1:k
    fx(j) = (fx(j) - fx(j-1))/(x(j)-x(j-k+1));
  end
end
```

```
function value = HornerNewton(a,x0,x)
% evaluate Newton polynom
% p(x0) = a(1) + a(2)*(x0-x(1)) + a(3)*(x0-x(1))*(x0-x(2)) + ...
%       = a(1) + ( (x0-x(1)) * ( a(2) + (x0-x(2)) * ( a(3) ....
value = a(end);
for k=length(a)-1:-1:1
  value = a(k) + value.*(x0-x(k));
end
```

MATLAB-Beispiel:

Testen wir das Differenzen-Verfahren nochmals an den beiden Beispielen aus Bsp. 2.2.3.1. Zum einen werten wir das Interpolationspolynom zu $f(x) = x^2$ und 3 Stützstellen an der Stelle 2 aus.

```
>> x=linspace(0,2,3);
>> a = NewtonInterpolation(x,x.^2)
a =
     0     1     1
>> HornerNewton(a,2,x)
ans =
     4
```

Zum anderen werten das Interpolationspolynom zu $f(x) = \sin(x)$ und 5 äquidistanten Stützstellen in $[0, 1]$ an der Stelle $\pi/3$ aus.

```
>> x=linspace(0,1,5);
>> a = NewtonInterpolation(x,sin(x));
>> sqrt(3)/2-HornerNewton(a,pi/3,x)
ans =
    4.387286117690792e-005
```

Beispiel 2.2.22 (*Hermite*-Interpolation via dem Schema der dividierten Differenzen)
Betrachten wir nun noch abschließend das Schema für die nichttriviale *Hermite*-Interpolationsaufgabe (d.h. auch Ableitungen werden interpoliert). Unter Beachtung von Lemma 2.2.19, insbesondere (2.11) ergibt sich:

$$\begin{array}{lllllll}
x_0: & [x_0]f & & & & & \\
 & & \searrow & & & & \\
x_0: & [x_0]f & \to & [x_0,x_0]f = f'(x_0) & & & \\
 & & \searrow & & \searrow & & \\
x_0: & [x_0]f & \to & [x_0,x_0]f = f'(x_0) & \to & [x_0,x_0,x_0]f = \tfrac{f''}{2}(x_0) & \\
 & & \searrow & & \searrow & & \searrow \\
x_1: & [x_1]f & \to & [x_0,x_1]f & \to & [x_0,x_0,x_1]f & \to \quad [x_0,x_0,x_0,x_1]f
\end{array}$$

Das resultierende Polynom $P(x)$ mit

$$P(x) = f(x_0) + (x - x_0)\Big(f'(x_0) + (x - x_0)\big(f''(x_0) + (x - x_0)[x_0, x_0, x_0x_1]f\big)\Big)$$

erfüllt dann die Interpolationsaufgaben

$$P(x_0) = f(x_0), P'(x_0) = f'(x_0), P''(x_0) = f''(x_0) \text{ und } P(x_1) = f(x_1).$$

Für den speziellen Fall, dass an allen Knoten $x_0, \ldots, x_n$ sowohl f als auch f' interpoliert werden sollen, erhält man die folgende Darstellung des Interpolationspolynoms $P(x)$.

Satz 2.2.23 (Darstellung des Interpolationspolynoms) *Es sei $\omega(x) = (x - x_0) \cdot \cdots \cdot (x - x_n)$ und $L_k^{(n)}(x)$ seien die* Lagrange*-Polynome zu den Knoten $x_0, \ldots, x_n$. Dann hat*

$$P(x) = \sum_{k=1}^{n} f(x_k)\left(1 - \frac{\omega''(x_k)}{\omega'(x_k)}(x - x_k)\right)L_k^{(n)}2(x) + \sum_{k=1}^{n} f'(x_k)(x - x_k)L_k^2(x)$$

die Interpolationseigenschaften

$$P(x_k) = f(x_k) \text{ und } P'(x_k) = f'(x_k), \quad k = 0, \ldots, n.$$

Beweis. Es sei x_ℓ einer der Knoten $x_0, \ldots, x_n$, dann folgt sofort aus der Interpolationseigenschaft der *Lagrange-Polynome*

$$P(x_\ell) = f(x_\ell),$$

da

$$\omega'(x) = \sum_{i=0}^{n} \prod_{\substack{k=0 \\ k\neq i}}^{n} (x - x_k) \text{ und somit } \omega'(x_\ell) = \prod_{\substack{k=0 \\ k\neq \ell}}^{n} (x_\ell - x_k) \neq 0$$

gilt. Für die Ableitung von P ergibt sich

$$\begin{aligned} P'(x) &= \sum_{k=1}^{n} f(x_k)\left[\Big(1 - \frac{\omega''(x_k)}{\omega'(x_k)}(x - x_k)\Big)2(L_k^{(n)}(x))' - \frac{\omega''(x_k)}{\omega'(x_k)} L_k^{(n)}(x)\right] L_k^{(n)}(x) \\ &\quad + \sum_{k=1}^{n} f'(x_k)\Big[(x - x_k)2(L_k^{(n)}(x))' + L_k^{(n)}(x)\Big] L_k^{(n)}(x), \end{aligned}$$

sodass wir nun Folgendes erhalten:

$$P'(x_\ell) = f(x_\ell)\Big(2(L_\ell^{(n)}(x_\ell))' - \frac{\omega''(x_\ell)}{\omega'(x_\ell)}\Big) + f'(x_\ell).$$

Nutzt man aus, dass $L_\ell^{(n)}(x) = \frac{\omega(x)}{(x-x_\ell)\omega'(x_\ell)}$ gilt (siehe Hausübung), so folgt aus

$$\omega(x) = L_\ell^{(n)}(x)(x - x_\ell)\omega'(x_\ell)$$

nach zweimaligem Differenzieren

$$\omega''(x) = (L_\ell^{(n)}(x))''(x - x_\ell)\omega'(x_\ell) + 2L_\ell'(x)\omega'(x_\ell).$$

Damit gilt an der Stelle x_ℓ

$$\frac{\omega''(x_\ell)}{\omega'(x_\ell)} = 2(L_\ell^{(n)}(x_\ell))''$$

Einsetzen in die Ableitung von P schließt den Beweis ab. □

2.2.4 Interpolationsgüte

Geht man davon aus, dass die Stützwerte f_i als Funktionswerte einer Funktion $f : [a, b] \to \mathbb{R}$ an den Stützstellen $x_i \in [a, b]$ gegeben sind, so stellt sich die Frage nach der Güte der Interpolation, d.h. wie gut approximiert das Interpolationspolynom die Funktion auf $[a, b]$.

2.2.4.1 Interpolationsfehler

Bei der nun folgenden Darstellung des Approximationsfehlers

$$\varepsilon(x) := f(x) - P(x)$$

erweisen sich die im vorherigen Abschnitt hergeleiten Eigenschaften der dividierten Differenzen als hilfreich:

Satz 2.2.24 (Interpolationsfehler) *Es sei $f \in C^n[a, b]$ und $f^{(n+1)}(x)$ existiere für alle $x \in (a, b)$; weiterhin gelte $a \leq x_0 \leq x_1 \leq \cdots \leq x_n \leq b$. Dann gilt:*

$$f(x) - P(f|x_0, \ldots, x_n)(x) = \frac{(x - x_0) \cdot \cdots \cdot (x - x_n)}{(n + 1)!} f^{(n+1)}(\xi), \tag{2.12}$$

wobei $\min\{x, x_0, \ldots, x_n\} < \xi < \max\{x, x_0, \ldots, x_n\}$ ist.

Beweis. Nach Konstruktion von $P(f|x_0,\dots,x_n)$ gilt:

$$P(f|x_0,\dots,x_n)(x_k) = f(x_k), \quad k = 0,1,\dots,n.$$

Es sei nun x fest und dabei ungleich $x_0, x_1, \dots, x_n$. Ferner sei

$$K(x) := \frac{f(x) - P(f|x_0,\dots,x_n)(x)}{(x-x_0)\cdot\dots\cdot(x-x_n)}. \tag{2.13}$$

Nun betrachten wir die Funktion

$$W(t) := f(t) - P(f|x_0,\dots,x_n)(t) - (t-x_0)\cdot\dots\cdot(t-x_n)K(x). \tag{2.14}$$

Die Funktion $W(t)$ verschwindet also an den Stellen $t = x_0, \dots, t = x_n$ und durch (2.13) auch an der Stelle $t = x$. Nach dem verallgemeinerten Satz von *Rolle*[3] verschwindet die Funktion $W^{(n+1)}(t)$ an einer Stelle ξ mit $\min\{x, x_0, \dots, x_n\} < \xi < \max\{x, x_0, \dots, x_n\}$. Das $(n+1)$-fache Differenzieren von (2.14) nach t liefert

$$W^{(n+1)}(t) = f^{(n+1)}(t) - (n+1)!K(x),$$

sodass gilt:

$$0 = W^{(n+1)}(\xi) = f^{(n+1)}(\xi) - (n+1)!K(x).$$

Damit erhalten wir aber unmittelbar

$$K(x) = \frac{1}{(n+1)!} f^{(n+1)}(\xi). \tag{2.15}$$

Nach Einsetzen von (2.15) in (2.14) erhalten wir die Behauptung für $t = x$, da x Nullstelle von W ist. □

Bemerkung 2.2.25 (Darstellung des Interpolationsfehlers) *Im Beweis zum Approximationsfehler (vgl. Satz, 2.2.24* (2.12)*) haben wir gezeigt*

$$f(x) - P(f|x_0,\dots,x_n)(x) = \frac{\omega_{n+1}(x)}{(n+1)!} f^{(n+1)}(\xi), \ \min\{x_0,\dots,x_n,x\} < \xi < \max\{x_0,\dots,x_n,x\}.$$

Und mit dem Satz über die Newton-*Darstellung gilt:*

$$f(x) - P(f|x_0,\dots,x_n)(x) = [x_0,\dots,x_n,x]f \cdot \omega_{n+1}(x).$$

Somit folgern wir: Für alle Knoten $x_0 \le \dots \le x_n$ existiert ein $\xi \in [x_0, x_n]$, sodass gilt:

$$[x_0,\dots,x_n]f = \frac{f^{(n)}(\xi)}{n!} \tag{2.16}$$

2.2.4.2 *Tschebyscheff*-Interpolation

Wie das folgende Beispiel zeigen wird, hat die Verteilung der Stützstellen $x_0, \dots, x_n$ über das Interpolationsintervall entscheidenden Einfluss auf die Güte der Approximation. Ein klassisches Beispiel hierfür stammt von *Runge*[4]:

Die Interpolationspolynome $P(f|x_1,\dots,x_n)$ zur Funktion $f(x) = \frac{1}{1+x^2}$ im Intervall $I := [-5,5]$ bei äquidistanten Stützstellen $x_k = -5 + \frac{10}{n}k$ zeigen bei wachsendem n einen zunehmenden Interpolationsfehler.

[3] Rolle, Michel (1652-1719)
[4] Runge, Carl (1856-1927)

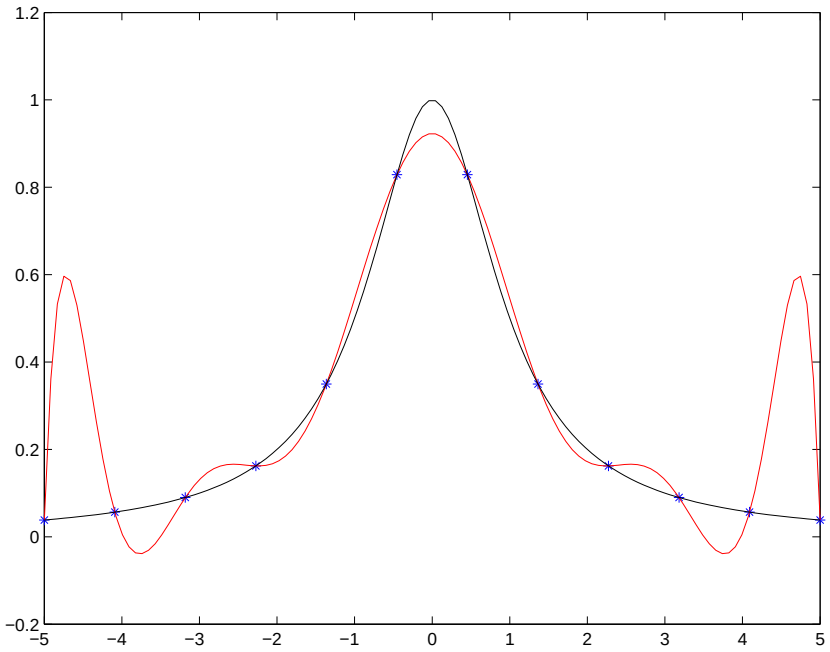
MATLAB-Beispiel:

Das Interpolationspolynom zu 12 äquidistanten Stützstellen und Stützwerten zur Funktion

$$f(x) = \frac{1}{1+x^2}$$

ist in Abb. 2.2.4.2 dargestellt.

```
n = 12;
f = @(x)  1./(x.^2+1);
x = linspace(-5,5,n);
fx = f(x);
s = linspace(x(1),x(end),10*n);
for j=1:length(s)
  ps(j) = AitkenNeville(x,fx,s(j));
end
plot(x,fx,'*',s,ps,'r-',s,f(s),'k')
```



Wie wir im weiteren zeigen werden, kann man bei geschickter, nichtäquidistanter Wahl der Stützstellen dagegen eine Konvergenz erhalten. Genauer gesagt, wählen wir $x_1, \ldots, x_n$ als die Nullstellen der von $[-1, 1]$ auf I transformierten ***Tschebyscheff*-Polynome** und erhalten dadurch punktweise Konvergenz für $n \to \infty$. Man beachte das dieses Phänomen nicht von Rundungsfehlern abhängt.

Bei der Berechnung des Approximations- bzw. des Interpolationsfehlers haben wir gesehen, dass

$$f(x) - P(f|x_0, \ldots, x_n)(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}\omega_{n+1}(x), \quad x \in [a, b]$$

für ein $\xi = \xi(x) \in (a, b)$ gilt. Wir suchen nun Knoten $x_0, \ldots, x_n \in [a, b]$, die das **Minimax-Problem**

$$\max_{x\in[a,b]} |\omega_{n+1}(x)| = \max_{x\in[a,b]} |(x - x_0) \cdot \ldots \cdot (x - x_n)| = \min$$

lösen. Anders formuliert, es gilt das normierte Polynom $\omega_{n+1} \in \mathbb{P}_{n+1}$ mit den reellen Nullstellen $x_0, \ldots, x_n$ zu bestimmen, für das

$$\max_{x\in[a,b]} |\omega_{n+1}(x)| = \min_{x_0,\ldots,x_n}$$

gilt. Im Folgenden werden wir sehen, dass gerade die *Tschebyscheff*-Polynome T_n diese obige *Minimax*-Aufgabe lösen (sie lösen sie bis auf einen skalaren Faktor und eine affine Transformation). Somit sind die Nullstellen der Tschebyscheff-Polynome (bis auf eine affine Transformation) gerade die gesuchten Stützstellen $x_0, \dots x_n$.

Zunächst reduzieren wir das Problem auf das Intervall $[-1, 1]$ mit Hilfe der Umkehrabbildung folgender Abbildung

$$y : [a, b] \to [-1, 1], \;\; x \mapsto y = y(x) = \frac{2x - a - b}{b - a},$$

d.h. die Umkehrabbildung lautet:

$$x : [-1, 1] \to [a, b], \;\; y \mapsto x = x(y) = \frac{b + a}{2} + \frac{b - a}{2} y.$$

Ist jetzt $P \in \mathbb{P}_n$ mit grad $P = n$ und führendem Koeffizienten $a_n = 1$ die Lösung des *Minimax*-problems

$$\max_{y \in [-1,1]} |P(x)| = \min,$$

so stellt $\widehat{P}(x) := P(y(x))$ die Lösung des ursprünglichen Problems mit führendem Koeffizienten $2^n/(b - a)^n$ dar. Für $y \in [-1, 1]$ definieren wir die ***Tschebyscheff*-Polynome** durch (vgl. hierzu auch Kapitel 3):

$$T_n(y) = \cos(n \arccos(y)), \;\; y \in [-1, 1] \tag{2.17}$$

und allgemein für $y \in \mathbb{R}$ durch die **Drei-Term-Rekursion**

$$T_0(y) = 1, \quad T_1(y) = y, \quad T_k(y) = 2yT_{k-1}(y) - T_{k-2}(y), \; k \geq 2. \tag{2.18}$$

Wir benötigen im Folgenden die Eigenschaften der *Tschebyscheff* Polynome, die wir in Kapitel 3 detailliert diskutierten werden und hier der Einfachheit halber bereits schon mal wiedergeben:

Bemerkung 2.2.26 (Eigenschaften der Tschebyscheff-Polynome) (i) *Der führende Koeffizient von T_n ist $a_n = 2^{n-1}$, $n \geq 1$*

(ii) $|T_n(y)| \leq 1$ *für* $y \in [-1, 1]$.

(iii) *Die Nullstellen von $T_n(y)$ sind*

$$y_k := \cos\left(\frac{2k + 1}{2n}\pi\right), \quad k = 0, 1, \dots, n - 1.$$

(iv) $|T_n(y)|$ *nimmt seinen maximalen Wert im Intervall* $[-1, 1]$ *an den* $(n + 1)$ *Extremalstellen* $\overline{y}_k = \cos(\frac{k\pi}{n})$ *für* $k = 0, \dots, n$ *an, d.h.*

$$|T_n(y)| = 1 \quad \Leftrightarrow \quad y = \overline{y}_k = \cos\left(\frac{k\pi}{n}\right) \;\; mit \; k = 0, \dots, n.$$

Satz 2.2.27 (Minimal-Eigenschaft der Tschebyscheff-Polynome) *Jedes Polynom $P \in \mathbb{P}_n$ mit führendem Koeffizienten $a_n \neq 0$ nimmt im Intervall $[-1, 1]$ einen Wert vom Betrag $\geq |a_n|/2^{n-1}$ an. Insbesondere sind die* Tschebyscheff*-Polynome $T_n(y)$ minimal bezüglich der Maximumsnorm $\|f\|_\infty = \max_{y \in [-1,1]} |f(y)|$ unter den Polynomen vom Grad n mit führendem Koeffizienten 2^{n-1}.*

Beweis. **(Annahme:)** Sei $P \in \mathbb{P}_n$ ein Polynom mit führendem Koeffizienten $a_n = 2^{n-1}$ und $|P(y)| < 1$ für $y \in [-1,1]$. Dann ist $T_n - P_n$ ein Polynom vom Grad kleiner oder gleich $(n-1)$ (beide besitzen a_n als führenden Koeffizienten). An den *Tschebyscheff*-Abszissen $\overline{y}_k := \cos(\frac{k\pi}{n})$ gilt:

$$T_n(\overline{y}_{2k}) = 1, \;\; P_n(\overline{y}_{2k}) < 1 \;\; \Rightarrow \quad P_n(\overline{y}_{2k}) - T_n(\overline{y}_{2k}) < 0,$$
$$T_n(\overline{y}_{2k+1}) = -1, \;\; P_n(\overline{y}_{2k+1}) > -1 \;\; \Rightarrow \quad P_n(\overline{y}_{2k+1}) - T_n(\overline{y}_{2k+1}) > 0,$$

d.h. die Differenz $T_n - P_n$ ist an den $(n+1)$-*Tschebyscheff*-Abszissen abwechselnd positiv und negativ, damit besitzt die Differenz mindestens n Nullstellen in $[-1,1]$ im Widerspruch zu $0 \neq T_n - P_n \in \mathbb{P}_{n+1}$. Demnach muss es für jedes Polynom $P \in \mathbb{P}_n$ mit führendem Koeffizienten $a_n = 2^{n-1}$ ein $y \in [-1,1]$ geben derart, dass $|P_n(y)| \geq 1$ erfüllt. Für ein beliebiges $P \in \mathbb{P}_n$ mit $a_n \neq 0$ folgt die Behauptung daraus, dass $\widetilde{P}_n := \frac{2^{n-1}}{a_n} P_n$ ein Polynom mit $\widetilde{a}_n = 2^{n-1}$ ist. □

2.3 Rationale Interpolation

Zur Interpolation einer Funktion, welche einen Pol besitzt oder deren Graph eine Asymptote aufweist sind im Allgemeinen Polynome nicht gut geeignet.

Zu diesem Zweck untersuchen wir im Folgenden die Interpolation mittels einer gebrochen rationalen Funktion

$$R(x) = \frac{p_0 + p_1 x + \ldots + p_\nu x^\nu}{q_0 + q_1 x + \ldots + q_\mu x^\mu} = \frac{P_\nu(x)}{Q_\mu(x)}. \tag{2.19}$$

Die Polynomgrade ν, μ der Polynome $P_\nu \in \mathbb{P}_\nu$ und $Q_\mu \in \mathbb{P}_\mu$ seien hierbei vorgegeben, sodass $R(x)$ an $(n+1)$ paarweise verschiedenen Knoten $x_0, \ldots, x_n$ die vorgegebenen Funktionswerte $f_0 := f(x_0), \ldots, f_n := f(x_n)$ annimmt, somit also gilt:

$$R(x_i) = f_i, \quad i = 0, \ldots, n.$$

Da Zähler und Nenner in (2.19) jeweils mit einer von Null verschiedenen Zahl multipliziert werden dürfen, enthält der Ansatz (2.19) $\nu + \mu + 1$ freie Unbekannte. Damit nun die Anzahl der Interpolationsbedingungen mit der Anzahl der Unbekannten $p_0, p_1, \ldots, p_\nu, q_0, q_1, \ldots, q_\mu$ übereinstimmt, muss zwangsläufig $\nu + \mu = n$ erfüllt sein.

Beispiel 2.3.1 $x_0 = 1, x_1 = 1, x_2 = 2, f_0 = 2, f_1 = 3, f_2 = 3$

Wir setzen $\nu = 0, \mu = 2$, d.h. wir verwenden den Ansatz

$$R(x) = \frac{p_0}{q_0 + q_1 x + q_2 x^2}.$$

Die Interpolationsaufgabe führt uns zum Gleichungssystem

$$P_\nu(x_i) - Q_\mu(x_i) \cdot f(x_i) = 0,$$

d.h. im vorliegenden Fall

$$\begin{aligned} p_0 - 2(q_0 - q_1 + q_2) &= 0 \\ p_0 - 2(q_0 + q_1 + q_2) &= 0 \\ p_0 - 3(q_0 + 2q_1 + 4q_2) &= 0 \end{aligned}$$

Nach der *Cramerschen* Regel besitzt dieses lineare Gleichungssystem die einzige Lösung

$$R(x) = \frac{36}{14 - 3x + x^2}.$$

Beispiel 2.3.2 Wählen wir zu den Daten von Beispiel 2.3.1 den Ansatz $\mu = \nu = 1$, so setzen wir

$$R(x) = \frac{p_0 + p_1 x}{q_0 + q_1 x},$$

bzw.

$$p_0 + p_1 x_i - f_i(q_0 + q_1 x_i) = 0, \quad i = 0, 1, 2$$

Dies führt uns zu folgendem linearen Gleichungssystem:

$$\begin{aligned} p_0 - p_1 - 2(q_0 - q_1) &= 0, \\ p_0 + p_1 - 3(q_0 + q_1) &= 0, \\ p_0 + 2p_1 - 3(q_0 + 2q_1) &= 0, \end{aligned}$$

welches die (bis auf einen gemeinsamen Faktor) eindeutige Lösung

$$p_0 = 3, \;\; p_1 = 3, \;\; q_0 = 1, \;\; q_1 = 1$$

besitzt, sodass wir folgendes Ergebnis erhalten:

$$R(x) = \frac{3 + 3x}{1 + x} = \tilde{R} = 3 \quad \text{[Widerspruch (vgl. } R(x_0) = 2\text{)]}$$

Bemerkung 2.3.3 *Die nichttriviale Lösung des LGS zur Bestimmung von $R(x)$ braucht nicht in jedem Fall die Interpolationsaufgabe zu erfüllen. Da sowohl Zähler als auch Nenner einen gemeinsamen Linearfaktor $(x - x_j)$ besitzen können, wird sich dieser bei der Lösung wegkürzen. Die daraus resultierende gebrochen rationale Funktion $\widetilde{R}(x)$ wird dann im Allgemeinen den Funktionswert f_j an der Stelle x_j nicht annehmen (vgl. hierzu Beispiel 2.3.2).*

Bemerkung 2.3.4 *Auch für die rationale Interpolation existieren* Neville*-artige Algorithmen zur Auswertung des Polynoms, ohne dies vorher bestimmt zu haben bzw. Algorithmen zur Bestimmung der Koeffizienten, ähnlich dem dividierten Differenzen Verfahren. Man schlage z.B. in [Stör/Bulirsch] unter den Stichworten* **Thielscher Kettenbruch**, **reziproke Differenzen** *nach.*

MATLAB-Funktion: RationalAitkenNeville.m

```
function value = RationalAitkenNeville(x,fx,x0)
% evaluate the rational interpolation polynomial (n,n+1) resp. (n,n
    )
% given by (x,fx) at the point x0
fxm1 = fx;
for j = length(fx):-1:2
  if x0~=x(j)
    fx(j) = fx(j) + (x0-x(j))/(x(j)-x(j-1)) * (fx(j)-fx(j-1));
  else
    value = fx(j); return
  end
end

for k = 3:length(fx)
  temp = fx;
  for j = length(fx):-1:k
    d = fx(j)-fx(j-1);
%    if fx(j)==fxm1(j-1) % abs(fx(j)-fxm1(j-1))<1e-10*max(1,max(abs
    (fx(j)),abs(fxm1(j-1))))
```

```
18 %       fx(j) = NaN;
19 %     else
20         %((x0-x(j-k+1))/(x0-x(j))*(1-d/(fx(j)-fxm1(j-1)))-1)
21         fx(j) = fx(j)+d/((x0-x(j-k+1))/(x0-x(j))*(1-d/(fx(j)-fxm1(j
               -1)))-1);
22 %     end
23     end
24     fxm1 = temp;
25 end
26 value = fx(end);
```

MATLAB-Beispiel:

Vergleichen wir die rationale Interpolation (Zähler- und Nennerpolynom quadratisch) mit der einfachen Polynominterpolation 4. Ordnung. Die Werte für $\cot(1°), \cot(2°), \ldots, \cot(5°)$ liegen vor und der Wert für $\cot(2°, 30')$ soll durch Interpolation angenähert werden.
Man beachte die unterschiedliche Genauigkeit.

```
>> format long
>> f = @(x) cot(x*pi/180);
>> x = linspace(1,5,5);
>> AitkenNeville(x,f(x),2.5)
ans =
  22.74709740132512
>> RationalAitkenNeville(x,f(x),2.5)
ans =
  22.90376554340344
>> f(2.5)
ans =
  22.90376554843120
```

Das die deutlich bessere Approximation durch das rationale Interpolationspolynom kein einzelner Zufallsbefund ist wird durch die folgende Grafik bestätigt, die den relativen Fehler zwischen Interpolationspolynom und exaktem Wert grafisch wiedergibt.

2.3.1 *Padé*-Approximation

Als *Padé*[5]-Approximation wird eine rationale Funktion bezeichnet, deren Potenzreihenentwicklung mit einer gegebenen Potenzreihe bis zum höchsten Grad übereinstimmt. Falls die rationale Funktion die folgende Form

$$R(x) = \frac{\sum_{k=0}^{m} a_k x^k}{1 + \sum_{k=0}^{n} b_k x^k}$$

hat, dann wird $R(x)$ als *Padé*-Approximation zur Potenzreihe

$$f(x) = \sum_{k=0}^{\infty} c_k x^k$$

bezeichnet, falls

$$R(0) = f(0) \quad \text{und ebenso} \tag{2.20}$$

$$\frac{d}{dx^k} R(x)\Big|_{x=0} = \frac{d}{dx^k} f(x)\Big|_{x=0}, \quad k = 1, 2, \ldots, m+n \tag{2.21}$$

gilt.

[5] Padé, Henri (1863-1953)

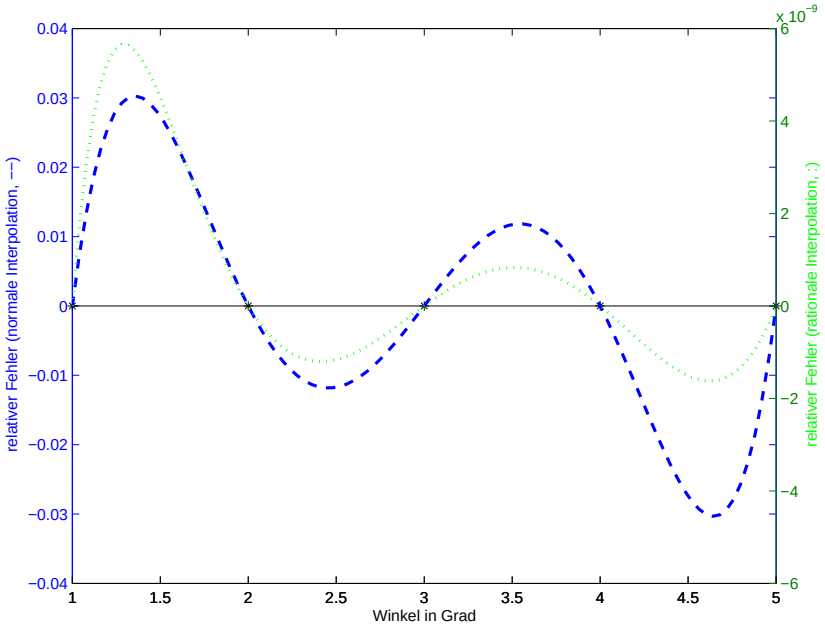


Abb. 2.1: Die Werte für $\cot(1°), \cot(2°), \ldots, \cot(5°)$ liegen vor. Dargestellt ist der relative Fehler zwischen Interpolationspolynom (zu den gegebenen Daten) und der exakten Funktion für das einfache Interpolationspolynom (gestichelte Linie, y-Achse links) mit einem Maximalwert ≈ 0.03 und für ein rationales Interpolationspolynom (gepunktete Linie, y-Achse rechts) mit einen maximalen Absolutwert $\approx 5.8 \cdot 10^{-9}$.

Die Gleichungen (2.20) und (3.43) führen zu $m + n + 1$ Gleichungen mit den Unbekannten $a_0, \ldots, a_m$ und $b_1, \ldots, b_n$. Der einfachste Weg zu diesen Gleichungen zu gelangen, besteht darin, $R(x)$ und $f(x)$ mit dem Nenner von $R(x)$ zu multiplizieren und einen Koeffizientenvergleich in den ersten $m + n + 1$ Monomen $1, x, x^2, \ldots, x^{m+n}$ durchzuführen.

Betrachten wir allerdings nur den Spezialfall $m = n$, so erhalten wir $a_0 = c_0$ und die Verbleibenden Koeffizienten a, b erfüllen die Gleichungen

$$\sum_{l=1}^{n} b_l c_{n-l+k} = -c_{n+k}, \quad k = 1, \ldots, n \tag{2.22}$$

$$\sum_{l=0}^{k} b_l c_{k-l} = a_k, \quad k = 1, \ldots, n \tag{2.23}$$

Man startet also zuerst mit (2.22), welches die Gleichungen zur Bestimmung von $b_1, \ldots, b_n$ darstellen. Obwohl diese Gleichungen eine *Töplitzmatrix* erzeugen, wendet man nicht das spezielle Verfahren für *Töplitzmatrizen* an. Erfahrungen zeigen, dass diese Gleichungen häufig nahezu singulär sind. Die LR-Zerlegung ist hier aus Stabilitätsgründen vorzuziehen. Nachdem die $b's$ nun bekannt sind, liefert (2.23) eine explizite Darstellung für die $a's$.

Die *Padé*-Approximation wird häufig dann angewendet, wenn die zu approximierende Funktion $f(x)$ nicht bekannt ist. Wir gehen davon aus, dass es möglich ist, den Wert von $f(x)$ und einiger Ableitungen an der Stelle $x = 0$, z.B. $f(0), f'(0), f''(0)$ zu bestimmen. Dies sind natürlich bereits die ersten Koeffizienten der Potenzreihenentwicklung, aber es ist keinesfalls klar, ob diese betragsmäßig kleiner werden und ob die Reihe überhaupt konvergiert. Wir wollen die Möglichkeiten der *Padé*-Approximation an einem Beispiel illustrieren, für eine Analyse der Methode verweisen wir auf [Cuyt/Wuytack].

Beispiel 2.3.5 Man stelle sich vor, man könnte die ersten fünf Terme einer Potenzreihe einer unbekannten Funktion $f(x)$ generieren, und diese sind

$$f(x) \approx 2 + \frac{1}{9}x + \frac{1}{81}x^2 - \frac{49}{8748}x^3 + \frac{175}{78732}x^4 + \dots$$

Wir wählen $m = n = 2$ für unsere *Padé*-Approximation. In der nachfolgenden Abbildung sind die abgebrochene Potenzreihendarstellung, die *Padé*-Approximation und die exakte Funktion

$$f(x) = \left(7 + (1+x)^{4/3}\right)^{1/3}$$

Die dargestellte Funktion f hat einen Verzweigungspunkt bei $x = -1$, daher konvergiert die Potenzreihe lediglich in $-1 < x < 1$. Im größten Teil der nachfolgenden Abbildung ist die Reihe divergent und der Wert der abgebrochenen Entwicklung nahezu sinnlos. Nichstdestoweniger liefert die *Padé*-Approximation sehr gute Werte zumindest bis $x \sim 10$.

MATLAB-Funktionen: pade.m und horner.m

```
function [a,b] = pade(c)
c=c(:); n=(length(c)-1)/2;
b =[1;-toeplitz(c(n+1:end-1),c(n+1:-1:2))\c(n+2:end)];
a = [toeplitz(c(1:n+1),[c(1),zeros(1,n)])*b]
```

```
function value = horner(a,x0)
value = a(end);
for k = length(a)-1:-1:1
  value = value.*x0 + a(k);
end
```

MATLAB-Beispiel:

Vergleich von abgebrochener Reihenentwicklung, *Padé*-Approximation und exakter Darstellung der Funktion

$$f(x) = \left(7 + (1+x)^{4/3}\right)^{1/3}.$$

Das Ergebnis ist in Abb. 2.3.1 dargestellt.

```
>> c = [2,1/9,1/81,-49/8748,175/78732];
>> x = linspace(0,10,400);
>> [a,b] = pade(c)
>> plot(x,(7+(1+x).^(4/3)).^(1/3),'k-', ...
     x,horner(c,x),'b-.', ...
     x,horner(a,x)./ horner(b,x),'r:')
>> axis([0,10,1.5,6])
>> legend('exakt','abgebr. Entwicklung',
   ...
     'Pade',2)
```

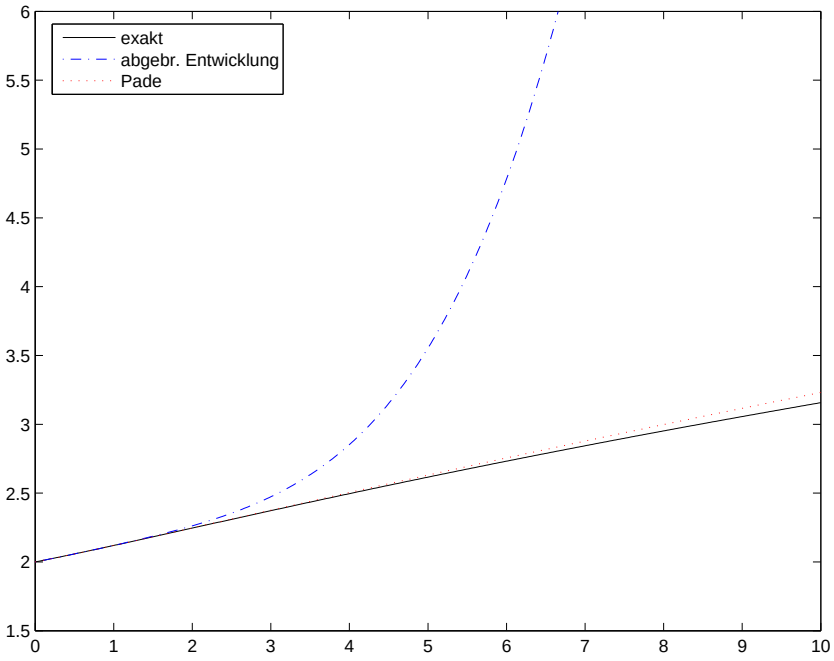


Abb. 2.2: Vergleich von abgebrochener Reihenentwicklung, *Padé*-Approximation und exakter Darstellung der Funktion $f(x) = (7 + (1 + x)^{4/3})^{1/3}$.

3 Numerische Integration und orthogonale Polynome

Die Numerische Integration oder Quadraturtheorie behandelt die Prinzipien und Algorithmen zur numerischen Berechnung von Integralen gegebener Funktionen. Hierbei beschränken wir uns vorerst auf die Berechnung des *Riemann*-Integrals[1]

$$I(f) := \int_a^b f(x)\, dx.$$

Dabei stellt f eine (von Computern ausführbare) Vorschrift dar, die es gestattet zu jedem $x \in [a,b]$ den entsprechenden Funktionswert f zu ermitteln (in den meisten Anwendungen ist f eine stückweise stetige oder stückweise glatte Funktion).

Aufgabe: Gegeben sei $f : [a,b] \to \mathbb{R}$, mit z.B. $f \in C[a,b]$. Approximiere den Ausdruck

$$I(f) := \int_a^b f(x)dx,$$

mit der Vorstellung, dass $I(f)$ nicht, bzw. nicht „leicht“ exakt bestimmt werden kann. Dazu konstruiert man Näherungsformeln (**„Quadraturformeln“**)

$$\hat{I}_n : C[a,b] \to \mathbb{R}, \quad f \to \hat{I}_n(f)$$

die das Integral möglichst gut approximieren und dessen Eigenschaften erhalten, so dass der **Quadraturfehler**

$$E_n(f) := I(f) - \hat{I}_n(f)$$

klein wird. Hierzu wird der Integrand durch eine Approximation, welche man z.B. durch Polynominterpolation erhält ersetzt und diese integriert.

Bevor wir diese Idee der Konstruktion von $\hat{I}_n$ vertiefen, Fragen wir uns zunächst, welche **Kondition** das Problem der Integralberechnung besitzt? Mit

$$\|f\|_1 := \int_a^b |f(t)|dt = I(|f|),$$

der so genannten L^1-Norm gilt

Lemma 3.0.1 (Kondition der Integration) *Die absolute und relative Kondition der Integralberechnung bzgl.* $\|\cdot\|_1$ *lauten:*

$$\kappa_{abs} = 1, \quad \kappa_{rel} = \frac{I(|f|)}{|I(f)|}. \tag{3.1}$$

Beweis. Sei $\delta f \in L_1([a,b])$ eine Störung, dann gilt

$$\left| \int_a^b (f+\delta f)dx - \int_a^b f\, dx \right| = \left| \int_a^b \delta f\, dx \right| \leq \int_a^b |\delta f| dx = \|\delta f\|_1$$

und hier gilt Gleichheit genau dann, wenn $\delta f \geq 0$. □

[1] Riemann, Georg Friedrich Bernhard (1826-1866)

D.h. in absoluter Betrachtung ist die Integration ein gutartiges Problem. Relativ betrachtet birgt die Integration z.B. von stark oszillierenden Integranten Schwierigkeiten.

3.1 QUADRATURFORMELN

Die **Grundidee** der Quadratur besteht aus dem Folgenden:

- Unterteile $[a, b]$ in m Teilintervalle $[t_k, t_{k+1}], k = 0, 1, \ldots, m-1$, z.B. äquidistant

$$t_k = a + kH,\ k = 0, \ldots, m,\ H = \frac{b-a}{m}.$$

- Approximiere den Integranden f auf jedem Teilintervall $[t_k, t_{k+1}]$ durch eine „einfach“ zu integrierende Funktion g_k (z.B. ein Polynom vom Grad kleiner gleich n bzgl. $(n+1)$ Stützstellen in dem Intervall $[t_k, t_{k+1}]$) und verwende die Approximation

$$I(f) = \sum_{k=0}^{m-1} \int_{t_k}^{t_{k+1}} f(y)\, dy \approx \sum_{k=0}^{m-1} \int_{t_k}^{t_{k+1}} g_k(y)\, dy.$$

  Für g_k verwenden wir zunächst die Polynominterpolation.

Wir beginnen in diesem Abschnitt mit einfachen Beispielen, bei denen wir jeweils zunächst nur ein Teilintervall betrachten auf welchem wir den Integranden mit einer „einfachen“ Funktion approximieren, dieser Ansatz wird dann jeweils für $m \geq 1$ Teilintervalle „zusammengesetzt“. Dadurch erhält man sog. zusammengesetzte oder summierte Regeln.

Insgesamt betrachten wir drei Beispiele; In den ersten zwei Beispielen verwenden wir auf jedem Teilintervall konstante Funktionen zur Approximation, d.h. konstante Interpolationspolynome nullten Grades bzgl. einer Stützstelle und im dritten Beispiel auf jedem Teilintervall lineare Funktionen, d.h. Interpolationspolynome ersten Grades bzgl. zweier Stützstellen.

Im Anschluss konstruieren dann so genannte interpolatorische Quadraturformeln und motivieren damit die allgemeine Definition von Quadraturformeln. Weiterhin werden wichtige Eigenschaften von Quadraturformeln behandelt.

3.1.1 Einfache Beispiele

Beginnen wir mit den sicherlich einfachsten Quadraturformeln. Allgemein ersetzen wir zunächst den Integranten f auf dem Intervall $[a, b]$ durch eine konstante Funktion.

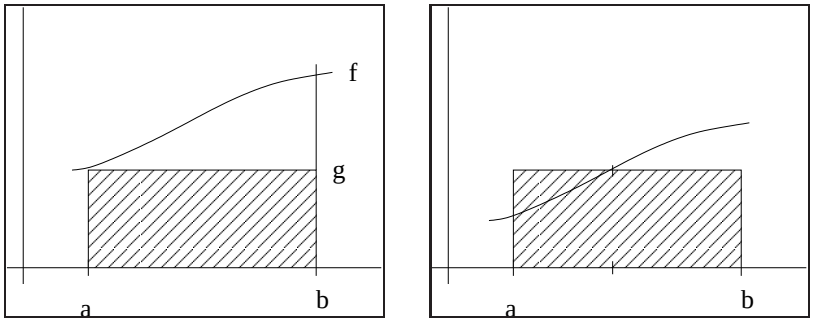


Abb. 3.1: Links: Linke Rechteckregel, rechts: Mittelpunktregel

Beispiel 3.1.1 (Linke Rechteckregel) Zunächst wählen wir als konstante Funktion g den Funktionswert am linken Rand, d.h. $x_0 = a$, $g(y) = f(x_0)$. Zur Veranschaulichung siehe Abbildung 3.1. Die resultierende Quadraturformel

$$R^L(f) := \hat{I}(f) = (b-a)f(a)$$

wird **linke Rechtecksregel** genannt. Natürlich könnte man auch analog den rechten Rand wählen.

Nun zerlegen wir das Integrationsintervall $[a,b]$ in $m \geq 1$ Teilintervalle $I_k = [t_k, t_{k+1}]$, $t_k = a + kH, k = 0, 1, \ldots, m-1$ und Breite $H = (b-a)/m$. Auf jedem Teilintervall wähle als konstante Funktion g_k den Funktionswert am linken Rand des Teilintervalls $g_k(y) := f(t_k)$, $y \in [t_k, t_{k+1}]$. Wir erhalten die Quadraturformel

$$R_m^L(f) := H \sum_{k=0}^{m-1} f(t_k),$$

diese wird **zusammengesetzte linke Rechtecksregel** genannt.

Nun zur Analyse: Sei f hinreichend glatt. Wegen

$$f(y) = f(t_k) + (y - t_k)f'(\xi)$$

mit einem $\xi \in (t_k, t_{k+1})$ (Taylor-Restglied) folgt

$$\begin{aligned}\int_{t_k}^{t_{k+1}} f(y)\,dy &= Hf(t_k) + \int_{t_k}^{t_{k+1}} f'(\xi)(y-t_k)\,dy \\ &= Hf(t_k) + \frac{f'(\xi)}{2}(t_{k+1}-t_k)^2 = \int_{t_k}^{t_{k+1}} g(y)\,dy + \mathcal{O}(H^2).\end{aligned}$$

Damit gilt

$$\begin{aligned}|I(f) - R_m^L(f)| &= \left| \int_a^b f(x)\,dx - H\sum_{k=0}^{m-1} f(t_k) \right| \\ &\leq \sum_{k=0}^{m-1} \underbrace{\left| \int_{t_k}^{t_{k+1}} f(x)\,dx - Hf(t_k) \right|}_{=\mathcal{O}(H^2)} \\ &\overset{m=\frac{b-a}{H}}{=} \mathcal{O}(H).\end{aligned}$$

Beispiel 3.1.2 (Mittelpunktregel) Wir setzen

$$\mathcal{C}_M^{(r)} := \left\{ f : [a,b] \to \mathbb{R} \;\middle|\; f^{(r)} \text{ stetig und } \sup_{a \leq x \leq b} |f^{(r)}(x)| \leq M \right\}$$

für $r \in \mathbb{N}$ und $M > 0$, und betrachten Grafik 3.2: Sei nun $x_0 \in (a,b)$, dann gilt für $f \in C^1[a,b]$:

$$\int_a^b |f - P(f|x_0)|\,dx \;\leq \sup_{a \leq x \leq b} \frac{|f'(x)|}{1!} \int_a^b |x - x_0|\,dx. \tag{3.2}$$

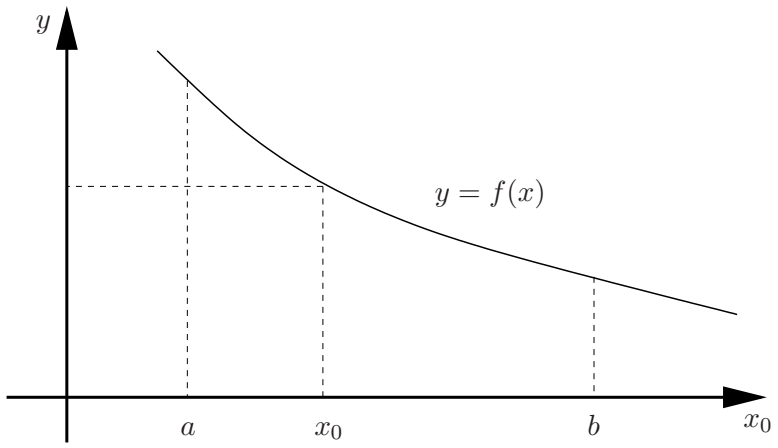


Abb. 3.2: Graph einer Funktion f.

Weiterhin ergibt sich mit $h := b - a$:

$$\begin{aligned} J(x_0) := \int_a^b |x - x_0|\, dx &= -\int_a^{x_0} (x - x_0)\, dx + \int_{x_0}^b (x - x_0)\, dx \\ &= \frac{(x-x_0)^2}{2}\Big|_{x=x_0}^b - \frac{(x-x_0)^2}{2}\Big|_{x=a}^{x_0} \\ &= \frac{(b-x_0)^2}{2} + \frac{(a-x_0)^2}{2} \\ &= x_0^2 - x_0(a+b) + \frac{a^2+b^2}{2}. \end{aligned}$$

Wir wollen die Stützstelle x_0 nun so wählen, dass $J(x_0)$ bzw. der rechte Teil der Ungleichung (3.2) minimal wird. Mit

$$J'(x_0) = 2x_0 - (a+b) \stackrel{!}{=} 0$$

ergibt sich als einziger kritischer Punkt $x_0 = \frac{a+b}{2}$, welcher J minimiert, da $J''(x_0) = 2 > 0$. Wir erhalten die **Mittelpunktregel** (zur Veranschaulichung siehe Abbildung 3.1)

$$M(f) := \hat{I}_0(f) := (b-a) \cdot f\left(\frac{a+b}{2}\right). \tag{3.3}$$

bei der als konstante Funktion $g(y) = f((a+b)/2)$ genommen wird.
Mit obiger Herleitung haben wir auch gezeigt, dass für $M := \sup_{a \le x \le b} |f'(x)|$ gilt:

$$\begin{aligned} \sup_{f \in \mathcal{C}_M^{(1)}} \left| \int_a^b f(x) - \hat{I}_0(f) \right| &\le \frac{M}{1!} \cdot J\Big(\frac{a+b}{2}\Big) \\ &= M \cdot \Big(\frac{(a+b)^2}{4} - \frac{(a+b)^2}{2} + \frac{a^2+b^2}{2}\Big) \\ &= M \cdot \frac{(b-a)^2}{4}, \end{aligned}$$

also die rechte Seite ein Maß für die von (3.3) in $\mathcal{C}_M^{(1)}$ garantierte Genauigkeit ist.

Nun zerlegen wir das Integrationsintervall $[a, b]$ wieder in $m \ge 1$ Teilintervalle $I_k = [t_k, t_{k+1}]$, $t_k = a + Hk, k = 0, \ldots, m$. Auf jedem Teilintervall setze

$$g_k(y) := f\left(\frac{t_k + t_{k+1}}{2}\right), \quad y \in [t_k, t_{k+1}].$$

Mit $x_k := \frac{1}{2}(t_k + t_{k+1})$ liefert Taylorentwicklung um x_k für $f \in C^3(t_k, t_{k+1})$

$$f(y) = f(x_k) + (y - x_k)f'(x_k) + \frac{1}{2}(y - x_k)^2 f''(x_k) + \mathcal{O}(H^3),$$

und damit

$$\begin{aligned}\int_{t_{k+1}}^{t_k} f(y)\,dy &= \int_{t_k}^{x_k} f(y)\,dy + \int_{x_k}^{t_{k+1}} f(y)\,dy \\ &= \frac{H}{2}f(x_k) + \int_{t_k}^{x_k}(y - x_k)f'(x_k)\,dy + \mathcal{O}(H^3) \\ &\quad + \frac{H}{2}f(x_k) + \int_{x_k}^{t_{k+1}}(y - x_k)f'(x_k)\,dy + \mathcal{O}(H^3) \\ &= Hf(x_k) + \mathcal{O}(H^3),\end{aligned}$$

denn

$$\int_{t_k}^{x_k}(y - x_k)\,dy = \int_{-\frac{H}{2}}^{0} z\,dz = -\int_{0}^{-\frac{H}{2}} z\,dz = -\int_{x_k}^{t_{k+1}}(y - x_k)\,dy.$$

Mit Quadraturknoten $x_k = a + (2k + 1)H/2, k = 0, \ldots, m - 1$ erhalten die **summierte Mittelpunktregel**

$$M_H(f) := \hat{I}_{0,m}(f) := H\sum_{k=0}^{m-1} f\left(\frac{t_k + t_{k+1}}{2}\right) = H\sum_{k=0}^{m-1} f(x_k),$$

mit dem Fehler

$$|I(f) - M_H(f)| = \mathcal{O}(H^2) \quad \text{für } f \in C^3(t_k, t_{k+1}).$$

Nun ersetzten wir den Integranten durch eine lineare Funktion bzw. durch lineare Funktionen auf Teilintervallen.

Beispiel 3.1.3 (Trapezregel) Wählen wir g als lineare Interpolation (Lagrange Interpolationspolynom vom Grad 1) von f bzgl. der Knoten $x_0 = a$ und $x_1 = b$ so erhalten wir die so genannte **Trapezregel**:

$$\hat{I}_1(f) := \frac{b - a}{2}\left(f(a) + f(b)\right).$$

Um die zusammengesetzte Trapezregel zu erhalten, zerlegen wir $[a, b]$ wie im Fall eines Knotens (d.h. $n = 0$) in $m \geq 1$ Teilintervalle $I_k = [t_k, t_{k+1}]$, $t_k = a + Hk, k = 0, \ldots, m$ der Breite $H = b - a/m$ und wähle g_k als lineare Interpolation von f bzgl. der Stützstellen t_k, t_{k+1} mit Quadraturknoten t_k, d.h.

$$g_k(y) = \frac{t_{k+1} - y}{H}f(t_k) + \frac{y - t_k}{H}f(t_{k+1}), \quad y \in [t_k, t_{k+1}]$$

Das führt zu

$$\int_{t_k}^{t_{k+1}} g_k(y)dy = \frac{H}{2}(f(t_{k+1}) + f(t_k)).$$

Dies setzt man zusammen auf $[a, b] = [t_0, t_m]$:

$$\hat{I}_{1,m}(f) := \frac{H}{2}\sum_{k=0}^{m-1}(f(t_{k+1}) + f(t_k))$$

Da jeder Term bis auf den ersten und letzten doppelt gezählt wird erhalten wir die sogenannte **Trapezsumme**

$$T_H(f) := \hat{I}_{1,m}(f) := H\left\{\frac{1}{2}f(a) + f(t_1) + \ldots + f(t_{m-1}) + \frac{1}{2}f(b)\right\}. \tag{3.4}$$

3.1.2 Konstruktion und Definition von Quadraturformeln

In obigen Beispielen haben wir stets den Integranden f durch eine Approximation $\hat{f}$ ersetzt, d.h. $f \approx \hat{f}$ und dann $\hat{f}$ exakt integriert. Wir haben also als Quadratur $\hat{I}_n$

$$\hat{I}_n(f) = I(\hat{f})$$

gewählt. Um höhere Genauigkeit zu erzielen, reichen konstante bzw. lineare Funktionen, wie bei der Mittelpunkts- bzw. der Trapetzregel, offenbar nicht aus.

Die Kernidee der Mittelpunkts- bzw. der Trapezformel liegt eigentlich darin, die Funktion f durch eine Interpolierende $P(f|x_0,\ldots,x_n)$ bzgl. der Stützstellen x_i (die - wie bei der Mittelpunktsregel - von den t_j verschieden seien können) zu ersetzen, sodass sich für diese die Quadratur einfach ausführen läßt. Es liegt somit nahe, als Approximation $\hat{f}$ das Interpolationspolynom an f bezüglich der Knoten zu wählen, d.h.

$$\hat{f}(x) = P(f|x_0,\ldots,x_n)(x).$$

Wir verwenden dann $I\big(P(f|x_0,\ldots,x_n)\big)$ als Approximation zu $I(f) = \int_a^b f(x)dx$, setzen also:

$$\hat{I}_n(f) = I\big(P(f|x_0,\ldots,x_n)\big).$$

Konstruktion interpolatorischer Quadraturformeln: Die Idee bei der Konstruktion von Quadraturformeln ist es $(n+1)$ Stützstellen zu wählen und, wie bereits erwähnt, das Integral über das Interpolationspolynom zu diesen Knoten als Näherung an das Integral von f zu verwenden.

Da ein besimmtes Integral über ein beliebiges Intervall $[a,b]$ - ohne Beschränkung der Allgemeinheit - mittels Variablentransformation auf das Intervall $[0,1]$ transformiert werden kann, betrachten wir zunächst $[a,b] = [0,1]$.
Die auf $[0,1]$ transformierten Stützstellen nennen wir τ_i.
Die $(n+1)$ Stützstellen $\tau_0,\ldots,\tau_n \in [0,1]$ seien der Einfachheit halber äquidistant, d.h. $\tau_i = ih$ für $i = 0,\ldots,n$ mit $h = 1/n$. Mit der Lagrange-Darstellung des Interpolationspolynoms

$$P(y) = \sum_{i=0}^{n} f(\tau_i) L_i^{(n)}(\tau), \quad L_i^{(n)}(y) = \prod_{\substack{j=0 \\ j\neq i}} \frac{(y-\tau_j)}{(\tau_i - \tau_j)}$$

folgt dann

$$\hat{I}_n(f) = \int_0^1 P(\tau)\,d\tau = \int_0^1 \left(\sum_{i=0}^{n} f(\tau_i) L_i^{(n)}(\tau)\right) d\tau = \sum_{i=0}^{n} f(\tau_i) \underbrace{\left(\int_0^1 L_i^{(n)}(\tau)\,d\tau\right)}_{=:\lambda_i} = \sum_{i=0}^{n} f(\tau_i)\lambda_i$$

Dies nennt man eine **Quadraturformel** mit **Gewichten** λ_i und **Knoten** (**Stützstellen**) τ_i.

Um eine allgemeine Quadraturformel für beliebige Intervalle $[a,b]$ zu bekommen, wird das Intervall $[a,b]$ nun auf $[0,1]$ transformiert. Mit $x = a + \tau(b-a)$ und $dx = d\tau(b-a) = (b-a)d\tau$ erhalten wir

$$I(f) = \int_a^b f(x)\,dx = (b-a)\int_0^1 \underbrace{f(a+\tau(b-a))}_{:=\tilde{f}(\tau)}\,d\tau.$$

Damit erhält man die Quadraturformel

$$\hat{I}_n(f) = (b-a)\sum_{i=0}^{n} f(a+\tau_i(b-a))\cdot\lambda_i$$

mit $\lambda_i := \int_0^1 L_i^{(n)}(\tau)\,d\tau$ und $\tau_i = ih$ für $h = 1/n$. Man beachte, die Gewichte λ_i sind unabhängig von den Integrationsgrenzen a, b und vom aktuellen Integranten und müssen für gegebene Stützstellen also nur *einmal* berechnet werden.

Wir fassen dies zusammen und definieren allgemein das lineare Funktional $\hat{I}_n$ als Quadraturformel:

Definition 3.1.4 (Quadraturformel) *Unter einer* Quadraturformel $\hat{I}$ *zur Approximation des bestimmten Integrals*

$$I(f) = \int_a^b f(x)\,dx$$

versteht man

$$\hat{I}_n(f) = (b-a)\sum_{i=0}^{n} \lambda_i f(x_i) \tag{3.5}$$

mit den Knoten $x_0, \ldots, x_n$ *und den* Gewichten $\lambda_0, \ldots, \lambda_n \in \mathbb{R}$*, wobei*

$$\sum_{i=0}^{n} \lambda_i = 1 \tag{3.6}$$

gilt.

Bemerkung 3.1.5

1. *Eine Eigenschaft wie (3.6) wird „Partition der Eins“genannt.*
2. *Sind die Gewichte nicht negativ, dann stellt (3.6) eine Konvexkombination der Funktionswerte $f(x_0), \ldots, f(x_n)$ dar.*
3. *Für die konstante Funktion $f(x) \equiv c = const$, $x \in [a, b]$ folgt aus (3.6)*

$$\hat{I}_n(f) = (b-a)\underbrace{\sum_{i=0}^{n} \lambda_i}_{=1} c = c(b-a) = I(f),$$

d.h., konstante Funktionen werden exakt *integriert. Dies ist der Grund für die Forderung (3.6).*

Definition 3.1.6 (Interpolatorische Quadraturformel) *Sei I ein Integral und I_n eine Quadraturformel dazu. Man nennt die Quaraturformel* **interpolatorisch***, wenn gilt*

$$\lambda_i = \frac{1}{(b-a)} I(L_i^{(n)}), \quad i = 0, \ldots, n.$$

Bemerkung 3.1.7 *Mit anderen Worten, im Rahmen der Herleitung der Quadraturformeln haben wir interpolatorische Quadraturformeln verwendet. Im Laufe dieses Kapitels werden wir noch andere Quadraturformeln kennen lernen, dies sind die so genannten* **Gauß-*Quadraturformeln*** *(vgl. Abschnitt 3.4).*

3.1.3 Exaktheitsgrad und Quadraturfehler

Aus der Konstruktion der interpolatorischen Quadraturformeln ergeben sich sofort Konsequenzen:

- derartige Quadraturformeln sind für alle $f \in \mathbb{P}_n$ exakt,
- aufgrund der Eindeutigkeit der Polynom-Interpolation ist diese Quadraturformel die *einzige* exakte bzgl. der Knoten $x_0, \ldots, x_n$.

Dies fassen wir in den folgenden Lemma zusammen:

Lemma 3.1.8 (Eindeutige exakte interpolatorische Quadraturformel) *Zu $(n+1)$ paarweise verschiedenen Knoten $x_0, \ldots, x_n$ gibt es genau eine Quadraturformel*

$$\hat{I}_n(f) = (b-a) \sum_{i=0}^{n} \lambda_i f(x_i), \tag{3.7}$$

die für alle $P \in \mathbb{P}_n$ vom Grad kleiner oder gleich n exakt ist.

Beweis. Wie bei der Konstruktion verwenden wird die *Lagrange*-Polynome $L_i^{(n)}$ setzen diese in die Quadraturformel ein und erhalten:

$$I\Big(\sum_{i=0}^{n} f(x_i) L_i^{(n)}(x)\Big) = \sum_{0=1}^{n} f(x_i) I\big(L_i^{(n)}(x)\big) = (b-a) \sum_{i=0}^{n} \lambda_i f(x_i). \tag{3.8}$$

Dadurch erhalten wir die Gewichte

$$\lambda_i = \frac{1}{(b-a)} \int_a^b L_i^{(n)}(x) dx$$

auf eindeutige Weise zurück. □

Nun untersuchen wir allgemein den Fehler von Quadraturformeln. Dazu definieren wir

Definition 3.1.9 (Quadraturfehler, Exaktheitsgrad, Fehlerordnung) *Sei I ein Integral, $\hat{I}_n$ eine Quadraturformel zu $(n+1)$ Knoten.*

1. *Dann heißt $E_n(f) := I(f) - \hat{I}_n(f)$ **Quadraturfehler**.*
2. *Die größte Zahl $r \in \mathbb{N}$ für die gilt*

   $$E_n(f) = I(f) - \hat{I}_n(f) = 0, \quad \forall f \in \mathbb{P}_r$$

   *wird **Exaktheitsgrad** der Quadraturformel genannt.*
3. *Zu gegebenem Exaktheitsgrad r wird $r+1$ **Ordnung** oder **Fehlerordnung der Quadraturformel** $\hat{I}_n$ genannt.*

Bemerkung 3.1.10 (Exaktheitsgrad interpolatorischer Quadraturformeln) *Nach Lemma 3.1.8 hat jede interpolatorische Quadraturformel, die $(n+1)$ Stützstellen verwendet, mindestens den Exaktheitsgrad n.*

Bemerkung 3.1.11 (Test der Ordnung einer Quadraturformel) *Der Quadraturfehler ist offenbar linear. Daher kann die Ordnung einer Quadraturformel $\hat{I}_n$ leicht über die Monome getestet werden: Gilt $E_n(x^j) = 0, j = 0, \ldots, r-1$ und $E_n(x^r) \neq 0$, so hat $\hat{I}_n$ die Ordnung r.*

Hat der Integrand f „schöne“ Eigenschaften, d.h. ist er z.B. $n+1$-mal stetig differenzierbar, so kann man den Interpolationsfehler mit Satz 2.2.24 abschätzen. Integration über den Interpolationsfehler liefert eine Fehlerschranke für den Integrationsfehler.

Lemma 3.1.12 (Fehlerschranken interpolatorischer Quadraturformeln) *Es gilt für*

$$\hat{I}_n(f) := \int_a^b P(f|x_0,\ldots,x_n)(x)dx$$

die Fehlerabschätzung

$$\begin{aligned} |E_n(f)| &\leq (b-a)\left(\max_{y\in[a,b]}\prod_{j=0}^{n}|y-x_j|\right)\frac{\|f^{(n+1)}\|_\infty}{(n+1)!} \\ &\leq \frac{(b-a)^{n+2}}{(n+1)!}\|f^{(n+1)}\|_\infty \end{aligned} \tag{3.9}$$

für $f \in C^{n+1}[a,b]$.

Beweis. Lagrange–Darstellung für $P(f|x_0,\ldots,x_n)$ einsetzen und Fehlerdarstellung der Polynominterpolation anwenden. □

Bemerkung 3.1.13 (Fehlerschranken) *Die Fehlerschranken aus Lemma 3.1.12 sind i.A.* nicht *optimal. D.h., die Schranken sind in der Regel zu groß. Schärfere Schranken können von Hand bewiesen werden (mittels Anwendung der Mittelwertsätze aus der Analysis).*

3.1.4 Konvergenz einer Quadraturformel

Das Ziel ist es, mit möglichst geringem Aufwand ein Integral möglichst genau zu approximieren. Als Maß für die Genauigkeit haben wir im letzten Abschnitt den Quadraturfehler eingeführt. Man kann sich zusätzlich noch fragen, ob, bzw. unter welchen Bedingungen Quadraturformeln, für wachsende Anzahl von Stützstellen gegen das exakte Integral konvergieren.

Wir betrachten dazu in diesem Abschnitt nun Folgen von Quadraturformeln $(\hat{I}_n)_{n\in\mathbb{N}}$ und untersuchen, wann diese gegen das zu approximierende Integral konvergieren. Zunächst definieren wir:

Definition 3.1.14 (Konvergenz einer Quadraturformel) *Sei $(\hat{I}_n)_{n\in\mathbb{N}}$ eine Folge von Quadraturformeln. Gilt*

$$\hat{I}_n(f) \xrightarrow{n\to\infty} \int_a^b f(x)dx \quad \text{für jedes } f \in C[a,b], \tag{3.10}$$

so spricht man von der **Konvergenz der Quadraturformeln** $(\hat{I}_n)_{n\in\mathbb{N}}$.

Die folgenden zwei Sätze liefern nun Bedingungen unter denen Folgen von Quadraturformeln $(\hat{I}_n)_{n\in\mathbb{N}}$ konvergieren. Insbesondere sind dies Bedingungen an die Gewichte.

Satz 3.1.15 (*Szegó*) *Sei $a \leq x_0^{(n)} < \cdots < x_n^{(n)} \leq b$, und seien $\lambda_k^{(n)} \in \mathbb{R}$, $k = 0,\ldots,n$. Dann sind die für $n \in \mathbb{N}$ gemäß*

$$\hat{I}_n : \big(C[a,b], \|\cdot\|_\infty\big) \to \big(\mathbb{R}, |\cdot|\big), \quad f \mapsto \hat{I}_n(f) := (b-a)\sum_{i=0}^{n}\lambda_i^{(n)} f\big(x_i^{(n)}\big)$$

definierten Quadraturformeln $\hat{I}_n$ genau dann konvergent, wenn die beiden folgenden Bedingungen erfüllt sind:

(i)

$$\sup_{n\in\mathbb{N}} \sum_{k=0}^{n} \left|\lambda_k^{(n)}\right| < \infty$$

(ii)

$$I(P) = \lim_{n\to\infty} \hat{I}_n(P), \quad \forall\, P \in \mathbb{P} = \mathbb{P}(a,b).$$

wobei $\mathbb{P}(a,b)$ die Menge der Polynome auf dem Intervall (a,b) bezeichnet.

Beweis. Der Beweis dieses Satzes erfordert grundlegende Kenntnisse der Funktionalanalysis und wird deshalb an dieser Stelle ausgespart. Man findet ihn nichtsdestoweniger beispielsweise in [Heuser, Seite 159]. □

Satz 3.1.16 (*Steklov*) *Unter den Voraussetzungen des Satzes von* Szegó *gelte*

$$\lambda_k^{(n)} \geq 0, \quad n \in \mathbb{N}, 0 \leq k \leq n \tag{3.11}$$

für die Gewichte $\lambda_k^{(n)}$ der Quadraturformel $\hat{I}_n$. Dann konvergieren die Quadraturformeln $(\hat{I}_n)_{n\in\mathbb{N}}$ genau dann, wenn sie für alle $P \in \mathbb{P}_n$ konvergieren.

Beweis. Nach dem Satz von *Szegó* ist der Beweis erbracht, wenn dort unter der Voraussetzung von (3.11) (i) aus (ii) folgt. Es gelte also (ii) im Satz von *Szegó*. Für $f \equiv 1$ gilt mit (ii)

$$b - a = I(f) = \lim_{n\to\infty} \hat{I}_n(f) = \lim_{n\to\infty} \sum_{k=0}^{n} \lambda_k^{(n)},$$

was wegen (3.11) die Aussage (i) impliziert. □

3.2 KLASSISCHE INTERPOLATORISCHE QUADRATURFORMELN

Wir haben Quadraturformeln über Lagrange-Interpolation zu äquidistanten Knoten konstruiert, d.h. durch Polynominterpolation. Damit haben wir die allgemeine Definition der Quadraturformeln motiviert. Im Fall, dass die Gewichte der Quadraturformel den Integralen über die Lagrange-Polynome entsprechen haben wir von interpolatorischen Quadraturformeln gesprochen.

Im Spezialfall, dass diese Quadraturformeln durch Polynominterpolation an *gleichverteilten* Knoten, d.h. äquidistanter Stützstellen - so wie wir sie auch konstruiert haben - , entstehen, spricht man von ***Newton-Cotes*[2]-Formeln**. In diesem Abschnitt schauen wir uns diesen Spezialfall noch mal etwas genauer an und fassen einige Eigenschaften zusammen. Wir werden sehen, dass die einführenden Beispiele zu diesen klassischen Quadraturformeln gehören.

Außerdem werden wir sehen, dass die Sätze aus dem vorherigen Abschnitt Grenzen für die numerische Stabilität dieses Ansatzes liefern. Dies führt uns auf die **zusammengesetzten *Newton-Cotes*-Formeln**.

[2] Cotes, Roger (1682-1716)

3.2.1 *Newton-Cotes*-Formeln

Definition 3.2.1 (*Newton-Cotes*-Formeln) *Bei äquidistanter Knotenwahl $a \leq x_0 < x_1 \cdots < x_n \leq b$ heißen die resultierenden Integrationsformeln*

$$\hat{I}_n(f) = (b-a)\sum_{i=1}^{n} \lambda_i f(x_i)$$

***Newton-Cotes-Formeln**.*
*Die Newton-Cotes-Formeln heißen **abgeschlossen**, wenn $x_0 = a$ und $x_n = b$, d.h.*

$$x_i = a + ih, \quad h = \frac{b-a}{n}, \quad i = 0, \ldots, n.$$

Die Newton-Cotes-*Formeln heißen **offen**, wenn $x_0 < a$ und $x_n < b$, wobei meist*

$$x_i = a + \Big(i + \frac{1}{2}\Big)h \quad \text{mit} \quad h = \frac{b-a}{n+1}, \quad i = 0, \ldots, n \tag{3.12}$$

gewählt wird.

Bemerkung 3.2.2 *Es ist klar, dass die Mittelpunkt- und die Trapezregel, welche wir bereits kennen gelernt haben, abgeschlossene* Newton-Cotes-*Formeln sind. Man erhält sie mit $n = 0, n = 1$.*

Der Ausdruck für die abgeschlossenen *Newton-Cotes*-Gewichte λ_i vereinfacht sich durch die Substitution $s := \frac{(x-a)}{h}$ zu

$$\lambda_i = \frac{1}{b-a}\int_a^b \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{(x-x_j)}{(x_i-x_j)}\, dx = \frac{1}{n}\int_0^n \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{(s-j)}{(i-j)}\, ds := \frac{1}{n}\alpha_i$$

und für die offenen *Newton-Cotes*-Formeln erhält man unter der Wahl (3.12) der x_i:

$$\lambda_i = \frac{1}{b-a}\int_a^b \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{(x-x_j)}{(x_i-x_j)}\, dx = \frac{1}{n+1}\int_{-\frac{1}{2}}^{n+\frac{1}{2}} \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{(s-j)}{(i-j)}\, ds := \frac{1}{n}\alpha_i$$

Bemerkung 3.2.3 (Praktische Berechnung via a priori Tabellierung der Gewichte) *Die* Newton-Cotes-*Formeln besitzen den Vorteil, dass die Gewichte zwar von n und h abhängen nicht aber von a, b. In der Praxis berechnet man daher* nicht *zunächst $\hat{f} = P(f|\ldots)$ und dann $I_n(\hat{f})$, sondern sucht direkt die Form (3.5) der Quadraturformel, d.h., man berechnet die Gewichte. Diese können* a priori *tabelliert werden.*

Bemerkung 3.2.4 (Offene *Newton-Cotes*-Formeln) *Die offenen* Newton-Cotes-*Formeln kann man vor allem dann verwenden, wenn die Auswertung des Integranden f an den Integrationsgrenzen schwierig ist, z.B. wenn f am Rand eine Singularität hat. Allerdings haben sie eine deutlich schlechtere Ordnung als die ebenfalls offenen Gauß-Quadraturen und finden daher in der Praxis kaum Anwendung.*

Im folgenden Beispiel tabellieren wir daher die Gewichte der wichtigsten geschlossenen *Newton-Cotes*-Formeln. Da die Stützstellen und Gewichte von dem Polynomgrad n abhängen, den wir wählen, schreiben wir hier, wie auch schon zuvor, $x_j^{(n)}$ und $\lambda_j^{(n)}$.

Beispiel 3.2.5 (Geschlossene *Newton-Cotes*-Formeln) Wählen wir $(n+1)$ äquidistante Stützstellen auf $[a, b]$, d.h.

$$x_i^{(n)} := a + \tau_i^{(n)}(b-a), \quad i = 0, \ldots, n,$$

wobei $\tau_i^{(n)} = i/n$, die Stützstellen auf $[0,1]$ sind. Bezeichne $\alpha_i^{(n)} = \lambda_i^{(n)} n$ die Gewichte auf $[0,1]$. Dann gilt mit $h = (b-a)/n$

$$\hat{I}_n(f) := (b-a)\sum_{i=0}^{n} \lambda_i^{(n)} f(a + \tau_i^{(n)}(b-a)) = h\sum_{i=0}^{n} \alpha_i^{(n)} f(a + \tau_i^{(n)}(b-a)). \qquad (3.13)$$

lauten die Eckdaten der wichtigsten geschlossenen *Newton-Cotes*-Formeln

n	$\tau_i^{(n)}$	$\alpha_i^{(n)}$	Restglied	Name
1	$0, 1$	$\frac{1}{2}, \frac{1}{2}$	$\frac{1}{12}h^3 f''(\xi)$	Trapezregel
2	$0, \frac{1}{2}, 1$	$\frac{1}{6}, \frac{2}{3}, \frac{1}{6}$	$\frac{1}{90}2^{-5}h^5 f^{(4)}(\xi)$	Simpson–Regel, Keplersche Fassregel
3	$0, \frac{1}{3}, \frac{2}{3}, 1$	$\frac{1}{8}, \frac{3}{8}, \frac{3}{8}, \frac{1}{8}$	$\frac{3}{80}3^{-5}h^5 f^{(4)}(\xi)$	Newtonsche $3/8$-Regel
4	$0, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}, 1$	$\frac{7}{90}, \frac{32}{90}, \frac{12}{90}, \frac{32}{90}, \frac{7}{90}$	$\frac{8}{945}4^{-7}h^7 f^{(6)}(\xi)$	Milne–Regel

Als Beispiel betrachte die Simpson-Regel, in obiger Form lautet diese

$$\hat{I}_2(f) = \frac{b-a}{2}\left(\frac{1}{6}f(a) + \frac{2}{3}f\left(\frac{a+b}{2}\right) + \frac{1}{6}f(b)\right).$$

Bemerkung 3.2.6 *In obigem Zugang entspricht die Anzahl der Stützstellen dem Exaktheitsgrad des Interpolationspolynoms und damit der Quadraturformel. Dies wiederum entspricht der Genauigkeit der Quadraturformel. Vielleicht kann man ja mit weniger Stützstelle dieselbe Genauigkeit erreiche, oder — anders ausgedrückt — mit geschickt gewählten Stützstellen eine höhere Genauigkeit erreichen. Mit dieser Frage wollen wir uns nun beschäftigen.*

Bemerkung 3.2.7 (Grenzen der *Newton-Cotes*-Formeln) *Bei den* Newton-Cotes-*Formeln treten durchaus negative Gewichte auf, d.h. der Satz 3.1.16 von* Steklov *kann dann nicht angewandt werden. Jedoch hilft auch der Satz 3.1.15 von* Szegó *nicht weiter, da - wie* G. Polya *zeigte - eine Funktion existiert, für die die* Newton-Cotes-*Formeln nicht konvergieren.*

Für abgeschlossene Newton-Cotes-*Formeln taucht bei $n = 8$, für offene* Newton-Cotes-*Formeln bei $n = 6$ das erste Mal ein negatives Gewicht auf. Man sollte die jeweiligen* Newton-Cotes-*Formeln nicht für n größer als diese jeweiligen Werte verwenden!*

Die Gewichte der Newton-Cotes-*Formeln sind* rational *und können mit* Maple *berechnet werden. Die folgenden Zeilen* Maple-*Code berechnen die Gewichte der abgeschlossenen* Newton-Cotes-*Formeln für $n = 8$ und die der offenen* Newton-Cotes-*Formeln für $n = 6$:*

MAPLE-*Beispiel:*

```
> n:=8:
> zaehler:= (x,i) -> quo(product((x-k),k=0..n),(x-i),x):
> seq(int(zaehler(x,m)/subs(x=m, zaehler(x,m)),x=0..n)/n,m=0..n);

   989    2944     -464    5248     -454     5248     -464     2944     989
 -----, ------, -------, ------, -----, ------, -------, ------, ------
 28350  14175    14175   14175    2835   14175    14175    14175    28350
```

```
> n:=6:
> zaehler:= (x,i) -> quo(product((x-k),k=0..n),(x-i),x):
> seq(int(zaehler(x,m)/subs(x=m, zaehler(x,m)),
      x=-1/2..n+1/2)/(n+1),m=0..n);

  4949    49    6223    -6257    6223    49    4949
 ------, ----, ------, ------, ------, ----, ------
 27648   7680  15360   34560    15360   7680  27648
```

Wie man sieht, enthalten beide Quadratur-Formeln negative Gewichte.

3.2.2 Zusammengesetzte *Newton-Cotes*-Formeln

Wir haben gesehen, dass bei den geschlossenen *Newton-Cotes*-Formeln ab $n = 7$ und für die offenen *Newton-Cotes*-Formeln ab $n = 5$ negative Gewichte auftreten. I.A. können wir keine Konvergenz $\hat{I}_n(f) \to I(f)$ für $n \to \infty$ erwarten.

Einen Ausweg bieten die sogenannten zusammengesetzten oder summierten Quadraturfomeln. Als Spezialfall zusammengesetzter *Newton-Cotes*-Formeln haben wir in den einführenden Beispielen bereits die summierte Mittelpunktsregel und die Trapezsumme kennen gelernt.

Allgemein ist die Idee zusammengesetzter Quadratur, dass man das Integrationsintervall $[a, b]$ in m Teilintervalle zerlegt und die Quadraturformel $\hat{I}_n(f)$ auf die Teilintervalle der Länge $H = (b-a)/m$ anwendet.

$$\hat{I}_{n,m}(f) = \sum_{k=0}^{m-1} \hat{I}_n|_{[t_k,t_{k+1}]}(f) = \sum_{k=0}^{m-1} \sum_{i=1}^{n} \lambda_i^{(n,k)} f(x_i^{(n,k)})$$

Sei nun n fest und man betrachte eine Folge von zusammengesetzte *Newton-Cotes*-Formeln $(\hat{I}_m(f))_{m\in\mathbb{N}}$:

Beispiel 3.2.8 (Anwendung des Satzes von *Szegó* auf die summierte Mittelpunktsregel)
Wir wenden den Satz von *Szegó* auf die summierte Mittelpunktsregel an. Betrachte hierzu eine Folge von Knoten

$$x_k^{(1,m)} := a + \frac{2k+1}{2} \cdot \frac{b-a}{m+1}, \quad k = 0, \ldots, m,$$

mit den Gewichten

$$\lambda_k^{(m)} = \frac{b-a}{m+1}.$$

Es gilt: $\sum_k \left|\lambda_k^{(m)}\right| = b - a$, demnach ist (i) in Satz 3.1.15 erfüllt. Weiterhin gilt (ii) aufgrund der Stetigkeit (damit der *Riemann*-Integrierbarkeit) der Polynome auf $[a, b]$ und der Fehlerabschätzung für die summierte Mittelpunktsregel.

Lemma 3.2.9 (Fehlerschätzung der Trapezsumme) *Es gilt*

$$|E_1(f)| = |I(f) - T_H(f)| = \mathcal{O}(H^2) \text{ für } f \in C^2([a, b]) \tag{3.14}$$

Bemerkung 3.2.10 (Ineffizenz bei hoher gewünschter Genauigkeit) *Angenommen, man möchte mit der Trapezsumme (3.4) die Genauigkeit von 10^{-6} erreichen. Die Fehlerschätzung (3.14) besagt dann*

$$|I(f) - T_H(f)| = \mathcal{O}(H^2) \leq 10^{-6},$$

also muss H^2 in der Größenordung von 10^{-6} sein, demnach muss $H \sim 10^{-3}$ gelten und bei äquidistanter Schrittweite folgt dann für die Anzahl der Knoten

$$m = \left[\frac{b-a}{H}\right] \sim 10^3.$$

Dies wäre viel zu ineffizient, wir brauchen Alternativen!

3.3 Extrapolation und Romberg-Integration

Wir wollen nun die Strategie zur nachträglichen Steigerung der Genauigkeit kennenlernen, die auch eine lokal angepasste („adaptive") Wahl der Stützstellen erlaubt. Bisher haben wir Quadraturen bzgl. eines festen Knoten gitters betrachtet. Nun verwenden wir eine Folge von Knotengittern.

Die Idee besteht darin zu einer Folge von absteigenden Schrittweiten verschiedene Approximationen an ein Integral zu berechnen, z.B. die Trapezsummen $T_{H_\ell}, \ell = 1, \ldots, k$. Dann wendet man Polynominterpolation zu den Paaren $(H_1, T_{H_1}), \ldots, (H_k, T_{H_k})$ an, um eine Interpolierende zu bestimmen. Wir wissen, dass

$$\lim_{H\to 0} I_H(f) = \lim_{m\to\infty} \hat{I}_{1,m}(f) = I(f)$$

gilt und werten daher das Polynom bei der optimalen Schrittweite $H = 0$ aus.

Da $H = 0$ außerhalb des gesicherten Bereichs $[H_1, H_k]$ liegt wird dieses Verfahren (welches auch in anderen Bereichen Anwendung findet) **Extrapolation** genannt. Prinzipiell lässt sich dieses Vorgehen auf beliebige Quadraturen anwenden, es eignet sich allerdings besonders gut für die Tapezsumme, von der es eine asymptotische Entwicklung des Fehlers gibt. Wir betrachten in diesem Abschnitt die summierte Trapezregel (3.4), welche wir einerseits bereits im Rahmen der einführenden Beispiele kennen gelernt haben und von der wir andererseits gesehen haben, dass es eine zusammengesetzte Newton-Cotes Formel ist. Zur Erinnerung (3.4) lautete:

$$T_H(f) = \hat{I}_{1,m}(f) = \frac{H}{2}\left\{f(a) + 2\sum_{i=1}^{m-1} f(x_i) + f(b)\right\}, \quad H := \frac{b-a}{m}.$$

Ausgangspunkt ist nun die folgende Aussage:

Lemma 3.3.1 (Asymptotische Entwicklung des Fehlers der Trapezsumme in H^2) *Sei $f \in C^{2\nu}[a,b]$. Der Fehler der Trapezsumme besitzt eine* asymptotische Entwicklung

$$T_H(f) - I(f) = \tau_2 H^2 + \tau_4 H^4 + \cdots + \tau_{2\nu} H^{2\nu} + \mathcal{O}(H^{2\nu+2}) \tag{3.15}$$

für $T_H(f)$ aus (3.4) mit Koeffizienten

$$\tau_{2\ell} = \frac{B_{2\ell}}{(2\ell)!}(f^{(2\ell-1)}(b) - f^{(2\ell-1)}(a)) \tag{3.16}$$

unabhängig von H und $B_{2\ell}$ sind die Bernoulli-Zahlen*, also*

$$B_0 = 1, \quad \sum_{\nu=0}^{n} \binom{n+1}{\nu} B_\nu = 0. \tag{3.17}$$

Beweis. Stoer, Taylor plus Symmetrie. □

Als Ansatz zur Erhöhung der Ordung wählt man eine Kombination aus zwei Trapezsummen zu unterschiedlichen Schrittweiten und hofft dadurch die Ordnung der asymptotischen Entwicklung des Fehlers zu erhöhen. Ersetzt man in (3.15) $T_H(f)$ z.B. durch

$$\tilde{T}_H(f) := \frac{4}{3}T_H(f) - \frac{1}{3}T_{2H}(f), \tag{3.18}$$

dann gilt

$$\begin{aligned}\tilde{T}_H(f) - I(f) &= \frac{4}{3}(T_H(f) - I(f)) - \frac{1}{3}(T_{2H}(f) - I(f)) \\ &= \frac{4}{3}\tau_2 H^2 + \frac{4}{3}\tau_4 H^4 + \cdots + \mathcal{O}(H^{2\nu+2}) \\ &\quad -\frac{1}{3}\tau_2 4H^2 - \frac{1}{3}\tau_4 16H^4 + \cdots + \mathcal{O}(H^{2\nu+2}) \\ &= \mathcal{O}(H^4).\end{aligned}$$

Durch die geschickte Kombination von $T_H(f)$ und $T_{2H}(f)$ fällt also der führende Term vor H^2 in der asymptotischen Entwicklung weg. Diese Idee kann man analog für allgemeine Schrittweiten H_1, H_2 anstelle von $H, 2H$, oder auch für andere Quadraturformeln realisieren.

Die Frage ist nun, wie macht man das praktisch, d.h., wie kommt man an geeignete Kombinationen wie in (3.18)? Eine Möglichkeit wäre, die entsprechenden Gleichungssysteme aufzustellen und zu lösen, dies ist jedoch unbequem, aufwendig und instabil für $H_1 \approx H_2$! Den allgemeinen Zugang nennt man **Extrapolation**. Extrapolation ist hierbei - wie bereits erwähnt - das Bestimmen eines Zusammenhangs über einen gesicherten Zusammenhang hinaus.

3.3.1 Idee der Extrapolation

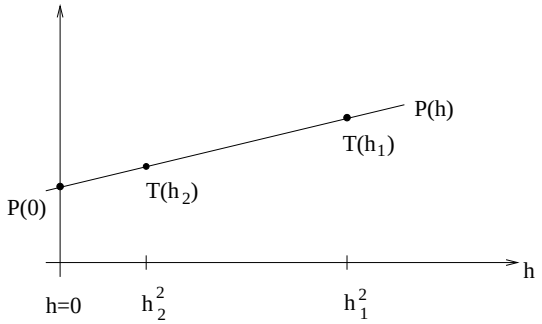
Die **Idee** lautet allgemeine wie folgt:

- seien H_1 und H_2 zwei Schrittweiten (für später $k = 2$)
- sei $P \in \mathbb{P}_{k-1} = \mathbb{P}_1$ ein Interpolationspolynom (Gerade) mit

$$P(H^2) = T_H(f) \text{ für } H = H_1, H_2, \tag{3.19}$$

  also

$$P(H^2) = T_{H_1}(f) + \frac{T_{H_2}(f) - T_{H_1}(f)}{H_2^2 - H_1^2}(H^2 - H_1^2)$$



- *extrapoliere* $P(H)$ auf $H = 0$, also

$$\begin{aligned} P(0) &= \frac{1}{H_2^2 - H_1^2}(T_{H_1}(f)(H_2^2 - H_1^2) - H_1^2(T_{H_2}(f) - T_{H_1}(f)) \\ &= (H_2^2 - H_1^2)^{-1}\left(H_2^2 T_{H_1}(f) - H_1^2 T_{H_2}(f)\right) \\ &= \mathcal{O}(H_1^2 H_2^2), \end{aligned}$$

  falls T_H die asymptotische Entwicklung (3.15) besitzt.

Um das entsprechende allgemeine Resultat allgemein zu beweisen, benötigen wir noch eine Hilfsaussage.

Lemma 3.3.2 *Für die Lagrange-Funktionen $L_i^{(n)}$ zu Stützstellen $x_0, \ldots, x_n$ gilt*

$$\sum_{j=0}^{n} L_j^{(n)}(0) x_j^m = \begin{cases} 1, & \text{für } m = 0, \\ 0, & \text{für } 1 \le m \le n, \\ (-1)^n x_0 \ldots x_n, & \text{für } m = n+1. \end{cases} \tag{3.20}$$

Beweis. Übung □

Nun haben wir alle Vorbereitungen abgeschlossen und können den Fehler abschätzen.

Satz 3.3.3 (Fehlerschätzung) *Sei $f \in C^{2k}[a,b]$ und $P \in \mathbb{P}_{k-1}$ mit $P(H_\ell^2) = T_{H_\ell}(f)$ für $\ell = 1, \ldots, k$, dann gilt*

$$\left| \int_a^b f(x)dx - P(0) \right| = \mathcal{O}(H_1^2 \cdots H_k^2). \tag{3.21}$$

Beweis. Verwende die Lagrange-Darstellung mit den Basis-Polynomen

$$L_i^{(k-1)}(H^2) = \prod_{\substack{\ell=1 \\ \ell \neq i+1}}^{k} \frac{H^2 - H_\ell^2}{H_{i+1}^2 - H_\ell^2}, \quad i = 0, \ldots, k-1,$$

dann hat das Interpolationspolynom die Gestalt

$$P(t) = \sum_{i=1}^{k} T_{H_i}(f) L_{i-1}^{(k-1)}(t),$$

also mit der asymptotischen Entwicklung und Lemma 3.3.2

$$\begin{aligned} P(0) &= \sum_{i=1}^{k} T_{H_i}(f) L_{i-1}^{(k-1)}(0) \\ &= \sum_{i=1}^{k} L_{i-1}^{(k-1)}(0) \left\{ I(f) + \sum_{\ell=1}^{k} \tau_{2\ell} H_i^{2\ell} + \mathcal{O}(H_i^{2k+2}) \right\} \\ &= I(f) \underbrace{\sum_{i=1}^{k} L_{i-1}^{(k-1)}(0)}_{=1} + \sum_{\ell=1}^{k} \tau_{2\ell} \underbrace{\sum_{i=1}^{k} L_{i-1}^{(k-1)}(0) H_i^{2\ell}}_{} + \underbrace{\sum_{i=1}^{k} L_{i-1}^{(k-1)}(0) \mathcal{O}(H_i^{2k+2})}_{\text{höhere Ordnung}} \\ &\qquad\qquad = \begin{cases} 0, & \ell \le k-1 \\ (-1)^{k-1} H_1^2 \cdots H_k^2, & \ell = k \end{cases} \\ &= I(f) + (-1)^{k-1} H_1^2 \ldots H_k^2 \tau_{2k} + \sum_{i=1}^{k} \mathcal{O}(H_i^{2k+2}). \end{aligned}$$

□

Dies kann man für k verschiedene Stützstellen und Verfahren der Ordung p zu einer **Rekursion** machen:

1. **Erzeuge Datenpaare:** Berechne $T_H(f)$ für k verschiedene Stützstellen $H_{i-k+1}, \dots, H_i$, $i = 1, 2, \dots$.

2. **Interpolation:** Bestimme das Interpolationspolynom in H^p:

$$P_{i,k}(H^p) = P(T_\cdot(f)|H^p_{i-k+1}, \dots, H^p_i)(H^p) \in \mathbb{P}_{k-1}(H^p)$$

   zu den Wertepaaren $(H^p_{i-j}, T_{H_{i-j}}(f)),\ j = 0, \dots, k-1$.

3. **Extrapolation:** $T_{i,k} := P_{i,k}(0)$, $1 \le k \le i$.

Zur Erinnerung, wir können das *Neville-Aitken*-Schema zur Auswertung eines Interpolationspolynoms verwenden:

$$P_{i,0} = f_i, \quad P_{i,k} = P_{i,k-1} + \frac{x - x_i}{x_i - x_{i-k}}(P_{i,k-1} - P_{i-1,k-1}), \quad i \ge k,$$

und dies überträgt sich dann für $T_{i,k}$ zu der folgenden **Rekursion**

$$\begin{aligned} T_{i,1} &:= T_{H_i}(f), && \text{für } i = 1, 2, \dots, \\ T_{i,k} &= T_{i,k-1} + \frac{T_{i,k-1} - T_{i-1,k-1}}{\left(\frac{H_{i-k+1}}{H_i}\right)^p - 1}, && 2 \le k \le i. \end{aligned}$$

Bemerkung 3.3.4 (Unterschied zum *Neville-Aitken*-Schema) *Man beachte, dass die vorstehende Rekursion im Gegensatz zum* Neville-Aitken-*Schema um einen Index verschoben ist. Das Extrapolationstablau, welches man analog zum Schema von* Neville *erhält, fängt erst beim Spaltenindex* 1 *an.*

Wir werden nun einen Algorithmus auf Basis obiger Rekursion formulieren: Geht man nun davon aus, dass eine Grundschrittweite H gegeben ist, und bildet die Schrittweiten H_i durch Teilen von H mit einer geeigneten Folge m_i. D.h. man erhält eine Schrittweitenfolge $(H_i)_{i\in\mathbb{N}}$ mit $H_i = H/m_i$, dann ergibt sich der folgende Algorithmus

Algorithmus 3.3.1 (*Romberg*[3]-Quadratur) *Gegeben sei eine Grundschrittweite H. Wähle eine Schrittweiten folge $H_1, H_2, \dots,$ mit $H_j = H/m_j, m_{j+1} > m_j$.*

Für $i = 1, 2, \cdots$:

1. *Bestimme $T_{i,1} := T_{H_i}(f)$.*
2. *Für $k = 2, \dots, i$: Berechne $T_{i,k} = T_{i,k-1} + \frac{T_{i,k-1} - T_{i-1,k-1}}{\left(\frac{m_i}{m_{i-k+1}}\right)^p - 1}$.*
3. *Falls $T_{i,i}$ ein geeignetes Abbruchkriterium erfüllt, STOPP.*

Bemerkung 3.3.5 (*Romberg*-Quadratur) *(a) Obiges Verfahren funktioniert für alle Quadratur-Verfahren $\hat{I}$ p-ter Ordnung mit entprechender asymptotischer Entwicklung.*

(b) Die Trapezregel hat eine asymptotische Entwicklung in H^2, wie wir in Lemma 3.3.1 gesehen haben, also $p = 2$, dieser Fall heißt **Romberg-*Quadratur***.*

(c) Die Schrittweiten-Folge

$$H_j = \frac{H}{m_j}, \quad m_{j+1} > m_j,$$

ist noch offen.

(d) Nach Satz 3.3.3 ist der Fehler $\mathcal{O}(H_1^p \cdots H_i^p)$.

(e) Der Aufwand A_i *zur Berechnung der* $T_{i,i}$ *hängt i.W. von der Anzahl der Auswertungen von* f *ab.*

In Algorithmus 3.3.1 wird weder genauer drauf eingegangen, was ein geeignetes Abbruchkriterium ist, noch erläutert, wie eine Schrittweitenfolge zu wählen ist. Für Letzteres werden wir im nun folgenden Abschnitt Beispiele kennen lernen.

3.3.2 Beispiele für Knotenfolgen

Wir betrachten nun Folgen $(m_i)_{i\in\mathbb{N}}$ und ordnen dieses Aufwandsfolgen

$$\mathcal{A} := \{A_1, A_2, \dots\}$$

zu, wobei

$$A_i = \text{Anzahl der zur Berechnung von } T_{i,i} \text{ notwendigen } f\text{-Auswertungen}$$

3.3.2.1 *Romberg*-Folge

Die folgende klassische ***Romberg*-Folge** führt zu einem einfachen Rechenschema

Definition 3.3.6 (Romberg-Folge) *Die Folge* $(m_i^R)_{i\in\mathbb{N}}$ *definiert durch*

$$m_i^R = 2^{i-1}, \quad \textit{d.h. } \{1, 2, 4, 8, 16, \dots\}. \tag{3.22}$$

wird **Romberg-*Folge*** *genannt.*

Der Vorteil ist die Möglichkeit der effizienten rekursiven Berechnung, also $A_i^R = m_i^R + 1$, d.h. $\{2, 3, 5, 9, 17, \dots\}$, wie folgende Skizze zeigt:

m_i^R	H_i	
1	1	2 Auswertungen
2	$1/2$	1 neue
4	$1/4$	2 neue

Beispiel 3.3.7 (*Romberg*-Folge) Für $H_1 = \frac{H}{m}$ betrachte die Trapezsumme

$$\begin{aligned} T_{\frac{H_1}{2}}(f) &= \frac{H_1}{4}\left\{f(a) + 2\sum_{i=1}^{2m-1} f\left(a + i\frac{H_1}{2}\right) + f(b)\right\} \\ &= \underbrace{\frac{H_1}{4}\left\{f(a) + 2\sum_{i=1}^{m-1} f(a + iH_1) + f(b)\right\}}_{=\frac{1}{2}T_{H_1}(f)} + \frac{H_1}{2}\sum_{i=1}^{m-1} f\left(a + \frac{2i-1}{2}H_1\right). \end{aligned}$$

Für die *Romberg*-Folge (3.22), d.h. $H_i = H \cdot 2^{1-i}$, gilt also

$$T_{i,1} = \frac{1}{2} T_{i-1,1} + H_i \sum_{j=1}^{m_{i-1}} f(a + (2j-1)H_i). \tag{3.23}$$

Bemerkung 3.3.8 (Stabilität bei Verwendung der *Romberg*-Folge) *Die* Romberg*-Folge führt auf eine Quadraturformel mit positiven Gewichten (*Übung*), was aus Stabilitätsgründen wichtig ist.*

3.3.2.2 Bulirsch-Folge

Vom Aufwand her günstiger als die *Romberg*-Folge ist die ***Bulirsch*-Folge**:

Definition 3.3.9 (Bulirsch-Folge) *Die Folge $(m_i^B)_{i\in\mathbb{N}}$ definiert durch*

$$1, 2, 3, 4, 6, 8, 12, 16, 24, \ldots, \quad m_i^B = \begin{cases} 2^j, & i = 2j, \\ 3 \cdot 2^{j-1}, & i = 2j+1, \\ 1, & i = 1, \end{cases} \tag{3.24}$$

mit $A_i = 2, 3, 5, 7, 9, 13, 17, \ldots$, d.h. $A_i^B = m_i^B + 1$, allerdings treten hier negative Gewichte auf. wird **Bulirsch-Folge** *genannt.*

Bemerkung 3.3.10 (Vorteil der Bulirsch-Folge) *Da $m_i^B < m_i^R$ für $i > 2$ gilt, folgt $A_i^B < A_i^R$ für $i > 2$, was zeigt, dass der Aufwand für die Bulirsch-Folge geringer als für die Romberg-Folge ist.*

Bemerkung 3.3.11 *(a) Die Verwendung einer asymptotischen Entwicklung setzt stillschweigend die entsprechende Glattheit von f voraus!*

(b) Wenn man f nicht kennt, oder f lokal stark variiert, ist es sinnvoll, die Schrittweite adaptiv *zu steuern, d.h.,*

- *große Schrittweiten in Bereichen, in denen f glatt ist,*
- *kleine Schrittweiten in Bereichen, in denen f variiert.*

Dazu benötigt man berechenbare so genannte Fehlerschätzer $\bar{\varepsilon}$ für den echten Fehler ε, d.h.

$$C_1 \varepsilon \leq \bar{\varepsilon} \leq C_2 \varepsilon.$$

Man kann zeigen, dass $|T_{k,k-1} - T_{k,k}|$ ein solcher ist, d.h. $|I(f) - T_{k,k}| \tilde{|} T_{k,k-1} - T_{k,k}|$ Das Prinzip lautet dann

- *$\bar{\varepsilon}$ „groß“: Unterteile das Intervall, verwende niedrige Ordnung,*
- *$\bar{\varepsilon}$ „klein “: verwende hohe Ordnung, vergröbere eventuell das Intervall.*

Details: [DH], 9.5-9.7.

3.4 *Gauß*-QUADRATUR

Wir haben gesehen, dass bei interpolatorische Quadraturen zu $(n+1)$ Stützstellen die Gewichte so gewählt werden können, dass diese mindestens den Exaktheitsgrad n haben. Bisher waren dabei die Stützstellen vorgegeben. Es stellt sich nun die Frage, ob wir Quadraturen mit höcherer Ordnung erreichen, wenn wir sowohl die Gewichte als auch Stützstellen frei wählen.

Wir verfolgen also das **Ziel**, Qudraturformeln zu konstruieren, die exakt sind von möglichst hoher Ordnung, d.h.

$$\hat{I}_n(P) = I(P)$$

für alle $P \in P_N$ mit N maximal (natürlich hängt N von n ab!).

Bevor wir uns allerdings mit der Konstruktion solcher Quadraturformel beschäftigen, d.h. wie mit der Wahl der Stützstellen und Gewichte, überlegen wir uns was N maximal konkret bedeutet. Hierzu halten wir zunächst das folgende fest:

Satz 3.4.1 (Obere Grenze für die Ordnung von Quadraturformeln) *Sind $a \le x_0^{(n)} < \cdots < x_n^{(n)} \le b$ und ist $\hat{I}_n$ eine Quadraturformel bzgl. $x_i^{(n)}, i = 0, \ldots, n$, so gilt für ihre Ordnung k:*

$$k \le 2n + 2.$$

Beweis. Die Anwendung der Quadraturformel in der Form

$$\hat{I}_n(f) = (b-a) \sum_{k=0}^{n} \lambda_k^{(n)} f\big(x_k^{(n)}\big)$$

auf das Polynom

$$P(x) = \prod_{k=0}^{n} \big(x - x_k^{(n)}\big)^2, \quad x \in [a, b]$$

vom Grad $2n + 2$ liefert die Aussage

$$\int_a^b \underbrace{P(x)}_{>0}\, dx - (b-a) \sum_{k=0}^{n} \lambda_k^{(n)} \underbrace{P\big(x_k^{(n)}\big)}_{=0} > 0,$$

also $I(P) > 0, \hat{I}_n(P) = 0$, speziell $I(P) \neq \hat{I}(P)$ und damit die Behauptung. □

Der vorstehende Satz besagt, dass für $(n + 1)$ Stützstellen Polynome vom Grad N kleiner gleich $2n + 1$ exakt integriert werden. Man kann sich leicht überlegen, dass dieser **maximale Exaktheitsgrad** auch erreicht wird:

Sei die Anzahl n der Teilintervalle vorgegeben. Die Quadraturformeln haben die Gestalt

$$\hat{I}_n(f) = (b-a) \sum_{i=0}^{n} \lambda_i^{(n)} f(x_i^{(n)})\, dx$$

Versucht man sowohl die $n + 1$ Stützstellen als auch die $n + 1$ Gewichte so zu wählen, dass die Fehlerordnung möglichst groß wird, so hat man $2n + 2$ Unbekannte:

$$\lambda_0^{(n)}, \ldots, \lambda_n^{(n)}, x_0^{(n)}, \ldots, x_n^{(n)},$$

Wenn man zur Bestimmung dieser Unbekannten den folgenden monomialen Ansatz wählt

$$(b-a) \sum_{i=0}^{n} \lambda_i^{(n)} (x_i^{(n)})^j = \int_a^b x^j\, dx = \frac{1}{j+1}(b^{j+1} - a^{j+1}), \quad j = 0, \ldots, 2n+1,$$

so ist dies ein **nichtlineares Gleichungssystem** mit $2n+2$ Gleichungen und $2n+2$ Unbekannten. Hat dieses Gleichungssystem eine Lösung, so integriert die resultierende Quadraturformel Polynome bis zum Grad $2n + 1$ exakt. Mit zunehmendem n wird dieses Gleichungssystem allerdings

immer unübersichtlicher. Und insbesondere gehen, wie bereits erwähnt, die Knoten nichtlinear ein.

Die Frage ist nur, wie man die Knoten und Gewichte systematischer wählen kann? Daher verfolgt man einen etwas allgemeineren Ansatz: Hierzu betrachten wir das gewichtete Integral

$$I(f) = \int_a^b f(x)\omega(x)\,dx$$

mit einer integrablen Gewichtsfunktion $\omega : [a, b] \to \mathbb{R}$. Wir nennen ω positiv, falls $\omega(x) > 0$ für alle $x \in (a, b)$.

Bevor wir uns aber mit der numerischen Integration eines solchen Integrals genauer beschäftigen, stellen wir noch einige mathematische Konstrukte zur Verfügung: Dies sind im Wesentlichen orthogonale Polynome und ihre Eigenschaften.

3.4.1 Orthogonale Polynome

In diesem Abschnitt werden orthogonale Polynome bzgl. gewichtetet L^2-Skalarprodukte definiert und ihre Eigenschaften betrachtet. Zunächst wiederholen wir die Definition eines Skalarprodukts:

Definition 3.4.2 (Skalarprodukt) *Es sei V ein reeller Vektorraum. Ein* **Skalarprodukt** *(oder inneres Produkt) auf V ist eine symmetrische positiv definite Bilinearform $(\cdot,\cdot)\colon V \times V \to \mathbb{R}$, d.h. für $x,\ y,\ z \in V$ und $\alpha,\ \beta \in \mathbb{R}$ gelten die folgenden Bedingungen:*

i.) positive Definitheit

$$(x, x) \geq 0, \quad \text{und } (x, x) = 0 \text{ genau dann, wenn } x = 0,$$

ii.) Symmetrie

$$(x, y) = (y, x)\,,$$

iii.) Bilinearität

$$(\alpha x + \beta y, z) = \alpha(x, z) + \beta(y, z)\,.$$

Man beachte, dass V hier nicht endlich-dimensional zu sein braucht! Für Skalarprodukte gilt die *Cauchy-Schwarz*-Ungleichung:

Satz 3.4.3 (*Cauchy-Schwarz*-Ungleichung) *Es sei V ein reeller Vektorraum, $(\cdot,\cdot)\colon V \times V \to \mathbb{R}$ ein Skalarprodukt auf V. Dann gilt*

$$(x, y) \leq \sqrt{(x, x)}\sqrt{(y, y)}\,.$$

Beweis. Der Fall $y = 0$ ist trivial. Es bleibt also der Fall $y \neq 0$ und damit $(y, y) \neq 0$. Für jedes $\alpha \in \mathbb{R}$ gilt

$$0 \leq (x - \alpha y, x - \alpha y) = (x, x) - 2\alpha(x, y) + \alpha^2(y, y)\,.$$

Wählt man nun speziell $\alpha := (x, y)/(y, y)$, so ergibt sich

$$0 \leq (x, x) - \frac{(x, y)^2}{(y, y)},$$

also

$$(x, y)^2 \leq (x, x)(y, y)\,.$$

Nun liefert Ziehen der Quadratwurzel die Behauptung. □

Mit Hilfe einer positiven, stetigen Gewichtsfunktion können wir nun das gewichtete L^2-Skalarprodukt einführen und definieren im Anschluss die Orthogonalität bzgl. dieses.

Satz 3.4.4 (Gewichtetes Skalarprodukt) *Es sei $\omega \in C(a,b)$, $\omega(x) > 0$ für $x \in (a,b)$ eine positive Gewichtsfunktion. Dann ist*

$$(f,g) := (f,g)_\omega := \int_a^b \omega(x) f(x) g(x)\, dx$$

für $f, g \in C[a,b]$ ein Skalarprodukt.

Beweis. Die Aussage ergibt sich unmittelbar aus der Verifizierung der bekannten Skalarprodukteigenschaften aus der Linearen Algebra (Additivität, Homogenität, Symmetrie, positive Definitheit) und wird deshalb an dieser Stelle ausgespart. □

Definition 3.4.5 (Orthogonalität) *Wir bezeichnen zwei Funktionen $f, g \in C[a,b]$ bzgl. des Skalarprodukts $(\cdot,\cdot)_\omega$ als* **orthogonal***, falls gilt:*

$$(f,g)_\omega = 0.$$

Bemerkung 3.4.6 (Norm und Momente zum gewichteten Skalarprodukt) *Im Folgenden setzen wir voraus, dass die durch das Skalarprodukt induzierte Norm*

$$\| Q \| := \| Q \|_\omega := \sqrt{(Q,Q)_\omega}$$

für alle Polynome $Q \in \mathbb{P}_n$, $n \in \mathbb{N}$ wohldefiniert und endlich ist. Somit existieren auch die Momente

$$m_n := \int_a^b \omega(x) x^n\, dx,$$

da mit der **Cauchy-Schwarz-*Ungleichung*** *folgt:*

$$|m_n| = |(1, x^n)_\omega| \le \|1\|_\omega \|x^n\|_\omega < \infty.$$

Beispiele 3.4.7 (Gewichtsfunktionen) Einige Beispiele von Gewichtsfunktionen seien hier genannt, die ungewöhnlich genug sind, um numerische Techniken zu erfordern, aber in praktischen Anwendungen verwendet werden.

- $\omega(x) = x^\alpha \log(1/x)$ auf $[0,1]$ mit $\alpha > 0$.

  Die Momente $m_n = (n+\alpha+1)^{-2}$ sind alle endlich und die zugehörigen Orthogonalpolynome werden verwendet, um Quadraturformeln zu konstruieren für Integrale über $[0,1]$, deren Integranden zwei Singularitäten bei Null haben, eine logarithmische und eine algebraische (falls $\alpha \neq 0, 1, 2, \ldots$).

- $\omega(x) = e^{-x}$ und $\omega(x) = e^{-x^2}$ auf $[0,c]$, mit $0 < c < \infty$.

  Dies sind *Laguerre* bzw. *Hermite*-Gewichte auf einem endlichen Intervall. Die Momente m_n lassen sich durch die unvollständige Gamma-Funktion $\gamma(\alpha, x) = \int_0^x t^{\alpha-1} e - t\, dt$ ausdrücken, nämlich $m_n = \gamma(n+1, c)$ bzw. $m_n = \frac{1}{2}\gamma(\frac{1}{2}(n-1), c^2)$. Beide Varianten finden Anwendung bei der Gauss-Quadratur von Integralen in der molekularen Quantenmechanik.

Definition 3.4.8 (Orthogonalpolynome) *Es sei $(P_n)_{n\in\mathbb{N}_0}$ eine Folge paarweise orthogonaler Polynome $P_n \in \mathbb{P}_n$, d.h.*

$$(P_i, P_j)_\omega = \delta_{ij} (P_i, P_i)_\omega \ge 0 \tag{3.25}$$

und exakt vom Grad n, dann heißen die P_n **Orthogonalpolynome** *über $[a,b]$ bzgl. der Gewichtsfunktion ω.*

Aufgabe 3.4.9 Man zeige, dass ein Polynom $P_n(x) = x^n + \ldots \in \mathbb{P}_n$ genau dann ein orthogonales Polynom n-ten Grades ist, wenn

$$\int P_n^2(x)\,\omega(x)\,dx = \min\left\{\int Q_n^2(x)\,\omega(x)\,dx : Q_n(x) = x^n + \ldots\right\}$$

gilt.

Bemerkung 3.4.10 *1. Da die $P_0, \ldots, P_n$ eine Basis des $\mathbb{P}_n$ darstellen, folgt sofort*

$$(P_n, Q)_\omega = (P_n, \sum_{i=0}^{n-1} \alpha_i P_i)_\omega = 0 \quad \text{für alle } Q \in \mathbb{P}_{n-1}\,. \tag{3.26}$$

2. Die Definition der Orthogonalpolynome über (3.25) ist keineswegs eindeutig. Eine mögliche Standardisierung ist z.B.

$$P_n(x) = x^n + \tilde{a}_{n-1}x^{n-1} + \ldots\,,$$

d.h. der führende Koeffizient ist Eins.

Definition 3.4.11 (Monisches Polynom) *Ist der führende Koeffizient eines Polynoms Eins, so bezeichnet man dieses als* **monisch** *(oder auch normiert).*

Der folgende Satz liefert nun das bzgl. jedes positiven gewichteten L^2-Skalarprodukts eindeutig bestimmte orthogonale momische Polynome existieren. Weiterhin lassen sich diese numerisch durch eine Rekursionsforschrift berechnen.

Satz 3.4.12 (Existenz und Eindeutigkeit monischer Orthogonalpolynome) *Zu jedem* positiv *gewichteten Skalarprodukt $(\cdot,\cdot)_\omega$ (also $\omega(x) > 0$) gibt es eindeutig bestimmte monische Orthogonalpolynome $P_n \in \mathbb{P}_n$ (d.h. mit führendem Koeffizienten Eins). Gilt zusätzlich $(x\,P_n, P_j)_\omega = (P_n, x\,P_j)_\omega$ für $j, n \geq 0$, dann erfüllen die Orthogonalpolynome die folgende Drei-Term-Rekursion:*

$$P_n(x) = (\alpha_n + x)P_{n-1}(x) + \gamma_n P_{n-2}(x), \quad n = 1, 2, \ldots \tag{3.27}$$

mit $P_{-1} := 0$, $P_0 := 1$ und

$$\alpha_n = -\frac{(xP_{n-1}, P_{n-1})_\omega}{(P_{n-1}, P_{n-1})_\omega}, \quad \gamma_n = -\frac{(P_{n-1}, P_{n-1})_\omega}{(P_{n-2}, P_{n-2})_\omega}. \tag{3.28}$$

Beweis. Man sieht unmittelbar, dass $P_0 \equiv 1 \in \mathbb{P}_0$ gilt, den Rest der Behauptung zeigen wir induktiv. Seien also $P_0, \ldots, P_{n-1}$ bereits bekannte paarweise orthogonale Polynome mit $P_j \in \mathbb{P}_j$ vom Grad j und führendem Koeffizienten gleich Eins. Aus diesen konstruieren wir nun P_n. Soll $P_n \in \mathbb{P}_n$ ebenso normiert sein, dann folgt zwangsläufig, dass

$$P_n(x) - xP_{n-1}(x) \quad \text{ist vom Grade } \leq n-1$$

und $P_0, \ldots, P_{n-1}, xP_{n-1}$ bilden eine Basis von $\mathbb{P}_n$. Da jedoch $P_0, \ldots, P_{n-1}$ eine Orthogonalbasis von $\mathbb{P}_{n-1}$ bzgl. $(\cdot,\cdot)_\omega$ bilden, gilt

$$P_n - xP_{n-1} = \sum_{j=0}^{n-1} \gamma_j P_j \;\text{ mit }\; \gamma_j := \frac{(P_n - xP_{n-1}, P_j)_\omega}{(P_j, P_j)_\omega}.$$

(Man setze die linke Gleichung in $(\cdot, P_\ell)_\omega$ für $\ell = 0, \ldots, n-1$ ein.) Außerdem folgt aus der Orthogonalität von P_n zu $P_0, \ldots, P_{n-1}$, dass nach Voraussetzung

$$\gamma_j = -\frac{(xP_{n-1}, P_j)_\omega}{(P_j, P_j)_\omega} = -\frac{(P_{n-1}, xP_j)_\omega}{(P_j, P_j)_\omega}\,.$$

Da $xP_j \in \mathbb{P}_{j+1}$, gilt somit $\gamma_j = 0$ für $j+1 < n-1$, d.h.

$$\gamma_0 = \cdots = \gamma_{n-3} = 0 \quad \text{bzw. es folgt} \quad P_n - xP_{n-1} = \gamma_{n-2}P_{n-2} + \gamma_{n-1}P_{n-1}\,.$$

Beachtet man $xP_{n-2} = P_{n-1} + \alpha_{n-2}P_{n-2} + \ldots + \alpha_0 P_0$, so folgt

$$\gamma_{n-2} = -\frac{(P_{n-1}, xP_{n-2})_\omega}{(P_{n-2}, P_{n-2})_\omega} = -\frac{(P_{n-1}, P_{n-1})_\omega}{(P_{n-2}, P_{n-2})_\omega}$$

und damit

$$\gamma_n = -\frac{(xP_n, P_n)_\omega}{(P_n, P_n)_\omega}\,.$$

Die Eindeutigkeit folgt induktiv mit Koeffizienten-Vergleich. Sei $Q_n \in \mathbb{P}_n$ eine weitere Folge orthogonaler Polynome. Wegen $Q_0 = P_0 = 1$ ist der Induktionsanfang klar. Weiter gilt

$$Q_n = \sum_{j=0}^{n} \alpha_j P_j \quad \text{mit} \quad \alpha_j = \frac{(Q_n, P_j)_\omega}{(P_j, P_j)_\omega}.$$

Aufgrund der Tatsache, dass der jeweilige führende Koeffizient eins ist, folgt $\alpha_n = 1$. Aus der Induktions-Voraussetzung ergibt sich $\alpha_0 = \cdots = \alpha_{n-1} = 0$, also $Q_n = P_n$. □

Bemerkung 3.4.13 *Die Abbildung*

$$(f, g) := -\int_{-1}^{1}\int_{-1}^{1} f(x)g(y)\log|x-y|\,dx\,dy$$

ist zwar ein Skalarprodukt auf der Menge der stetigen Funktionen, aber die zugehörigen Orthogonalpolynome erfüllen keine Drei-Term-Rekursion. Symmetrieüberlegungen zeigen sofort, dass für $0 < j + k$ *ungerade*

$$(x^j, x^k) = 0$$

gilt. Mit Hilfe von Maple verifiziert man leicht durch mehrfaches Anwenden der Anweisungen

```
j:=0;
k:=1;
-int(int(x^j*y^k*log((x-y)^2)/2,x=-1..1),y=-1..1);
```

zu verschiedenen $j, k \geq 0$*, dass gilt*

$$(x^j, x^k) = \begin{cases} 0 & \text{, falls } 0 \leq k+j \text{ ungerade,} \\ \neq 0 & \text{, falls } 0 \leq k+j \text{ gerade.} \end{cases}$$

Wir definieren $P_0 = 1$*,* $P_1 = x$ *und normieren weitere* P_k *so, dass der führende Koeffizient 1 ist. Damit das Orthogonalpolynom* $P_2 = x^2 + ax + b \in \mathbb{P}_2$ *orthogonal zu* x *ist, muss* $a = 0$ *gelten und aus* $(P_2, P_0) = 0$ *folgt* $b = -(x^2, 1)/(1, 1)$*. Man beachte, dass hier*

$$16/9 - 4/3\log(2) = (x^2, 1) \neq (x, x) = 1$$

gilt und somit die Formel der Drei-Term-Rekursion (3.27) *nicht gilt.*

Generell sind verschiedene Formen der Normierung der Orthogonalpolynome möglich. Wir führen die **allgemeine Darstellung der Orthogonalpolynome** in der Form

$$P_n(x) = k_n x^n + k'_n x^{n-1} + \ldots, \quad n = 0, 1, 2 \ldots\,. \tag{3.29}$$

ein. Zusätzlich definieren wir

$$h_n(P_n) := h_n := \int_a^b \omega(x) P_n^2(x)dx = \|P_n\|_\omega^2\,. \tag{3.30}$$

Bemerkung 3.4.14 (Normierung der Orthogonalpolynome) *Bezüglich der allgemeinen Darstellung (3.29) und (3.30) sind Polynome* ***orthonormal****, wenn* $h_n(P_n) = 1$ *und* ***monisch****, falls* $k_n = 1$ *gilt .*

Übertragen wir nun die Drei-Term-Rekursion aus Satz 3.4.12 für Orthogonalpolynome mit führendem Koeffizienten Eins auf die allgemeinere Form (3.29), so erhalten wir folgendes Resultat. Hierbei kann man die Koeffizienten der Rekursionsformel aus den Leittermen der monischen Polynomen ablesen.

Satz 3.4.15 (Existenz und Eindeutigkeit allgemeiner orthogonaler Polynome) *Zu jedem Skalarprodukt* $(\cdot,\cdot)_\omega$ *gibt es eindeutig bestimmte Orthogonalpolynome* $P_n \in \mathbb{P}_n$ *mit führendem Koeffizienten* k_n*. Diese Polynome erfüllen die folgende Drei-Term-Rekursion:*

$$P_n(x) = (a_n + b_n x)P_{n-1} + c_n P_{n-2}, \quad n = 1, 2, \ldots, \tag{3.31}$$

mit $P_{-1} = 0$*,* $P_0 = k_0$*. Die Koeffizienten ergeben sich wie folgt:*

$$\begin{aligned} b_n &= \frac{k_n}{k_{n-1}}, \qquad a_n = b_n\left(\frac{k_n'}{k_n} - \frac{k_{n-1}'}{k_{n-1}}\right), \quad n = 1, 2, \ldots, \\ c_1 &= \textit{bel.} < \infty, \quad c_n = -\frac{k_n k_{n-2} h_{n-1}}{k_{n-1}^2 h_{n-2}}, \quad n = 2, 3, \ldots . \end{aligned} \tag{3.32}$$

Beweis. Die Behauptung folgt aus Satz 3.4.12 mit der Darstellung (3.29). Sei $P_n(x) = k_n x^n + k_n' x^{n-1} + \ldots, n = 0, 1, 2\ldots$, dann hat $\bar{P}_n(x) := P_n(x)/k_n$ führenden Koeffizienten Eins und Satz 3.4.12 liefert

$$P_n(x) = k_n \bar{P}_n(x) = k_n(\alpha_n + x)\bar{P}_{n-1}(x) + k_n\gamma_n\bar{P}_{n-2}(x), \quad n = 1, 2, \ldots \tag{3.33}$$

mit $\bar{P}_{-1} := 0$, $\bar{P}_0 := P_0/k_0 = 1$ und

$$\alpha_n = -\frac{(x\bar{P}_{n-1}, \bar{P}_{n-1})_\omega}{(\bar{P}_{n-1}, \bar{P}_{n-1})_\omega}$$

sowie

$$\gamma_n = -\frac{(\bar{P}_{n-1}, \bar{P}_{n-1})_\omega}{(\bar{P}_{n-2}, \bar{P}_{n-2})_\omega} = -\frac{1/k_{n-1}^2\,(P_{n-1}, P_{n-1})_\omega}{1/k_{n-2}^2\,(P_{n-2}, P_{n-2})_\omega} = -\frac{k_{n-2}^2}{k_{n-1}^2}\frac{h_{n-1}}{h_{n-2}}\,. \tag{3.34}$$

Man beachte

$$x\bar{P}_{n-1} = x\left(x^{n-1} + \frac{k_{n-1}'}{k_{n-1}}x^{n-2} + \ldots\right) = \bar{P}_n + \left(\frac{k_{n-1}'}{k_{n-1}} - \frac{k_n'}{k_n}\right)\bar{P}_{n-1} + Q(x) \quad \text{mit } Q \in \mathbb{P}_{n-2}\,,$$

d.h.

$$\alpha_n = -\frac{(x\bar{P}_{n-1}, \bar{P}_{n-1})_\omega}{(\bar{P}_{n-1}, \bar{P}_{n-1})_\omega} = \frac{k_n'}{k_n} - \frac{k_{n-1}'}{k_{n-1}}\,. \tag{3.35}$$

Somit ergibt sich mit (3.34)

$$k_n\gamma_n\bar{P}_{n-2}(x) = -k_n\frac{k_{n-2}^2}{k_{n-1}^2}\frac{h_{n-1}}{h_{n-2}}\frac{1}{k_{n-2}}P_{n-2}(x) = -\frac{k_n k_{n-2}}{k_{n-1}^2}\frac{h_{n-1}}{h_{n-2}}P_{n-2}(x)$$

bzw.

$$k_n\, x\, \bar{P}_{n-1}(x) = \frac{k_n}{k_{n-1}}\, x\, P_{n-1}(x)$$

und mit (3.35)

$$k_n\alpha_n\bar{p}_{n-1}(x) = \frac{k_n}{k_{n-1}}\left(\frac{k_n'}{k_n} - \frac{k_{n-1}'}{k_{n-1}}\right)P_{n-1}(x)\,.$$

Die letzten drei Gleichungen mit (3.4.1) liefern dann die Behauptung. □

Satz 3.4.16 (Summenformel von *Christoffel-Darboux*) *Mit der für Orthogonalpolynome zuvor eingeführten Notation* (3.29), (3.30) *gilt die Summenformel von* Christoffel[4]-Darboux[5]

$$\sum_{i=0}^{n} \frac{P_i(x)P_i(y)}{h_i} = \frac{k_n}{k_{n+1}\cdot h_n} \cdot \frac{P_{n+1}(x)P_n(y) - P_n(x)P_{n+1}(y)}{x-y}. \tag{3.36}$$

Beweis. Für $n = 0$ erhält man die Identität (3.36) durch Einsetzen von $P_0(x) = k_0$ und $P_1(x) = k_1 x + k_1'$ und für $n \geq 1$ aus der Rekursionsformel (3.31). Aus dieser folgt nämlich für $n \geq 1$ durch jeweiliges Einsetzen für P_{n+1}

$$\begin{aligned} & P_{n+1}(x)P_n(y) - P_n(x)P_{n+1}(y) \\ &= \big((a_{n+1} + b_{n+1}x)P_n(x) + c_{n+1}\,P_{n-1}(x)\big)P_n(y) \\ &\quad - P_n(x)\big((a_{n+1} + b_{n+1}\,y)P_n(y) + c_{n+1}\,P_{n-1}(y)\big) \\ &= b_{n+1}(x-y)P_n(x)P_n(y) + c_{n+1}\big(P_{n-1}(x)P_n(y) - P_n(x)P_{n-1}(y)\big) \end{aligned}$$

Mit (3.32) erhalten wir für $x \neq y$ ($b_{n+1} = k_{n+1}/k_n \neq 0$ nach Voraussetzung)

$$\begin{aligned} & \frac{1}{b_{n+1}} \cdot \frac{P_{n+1}(x)P_n(y) - P_n(x)P_{n+1}(y)}{x-y} \\ &= P_n(x)P_n(y) + \frac{c_{n+1}}{b_{n+1}} \cdot \frac{P_{n-1}(x)P_n(y) - P_n(x)P_{n-1}(y)}{x-y}, \end{aligned}$$

d.h.

$$\begin{aligned} & \frac{k_n}{k_{n+1}} \cdot \frac{P_{n+1}(x)P_n(y) - P_n(x)P_{n+1}(y)}{x-y} \\ = \quad & P_n(x)P_n(y) + \frac{k_{n+1}k_{n-1}h_n k_n}{k_n^2\, h_{n-1}\, k_{n+1}} \cdot \frac{P_n(x)P_{n-1}(y) - P_{n-1}(x)P_n(y)}{x-y} \\ = \quad & P_n(x)P_n(y) + \frac{k_{n-1}h_n}{k_n h_{n-1}} \cdot \frac{P_n(x)P_{n-1}(y) - P_{n-1}(x)P_n(y)}{x-y}. \end{aligned} \tag{3.37}$$

Gelte nun (3.36) für ein $n-1 \geq 0$ dann schließen wir mit (3.37) auf n wie folgt

$$\begin{aligned} & \frac{k_n}{k_{n+1}} \cdot \frac{P_{n+1}(x)P_n(y) - P_n(x)P_{n+1}(y)}{x-y} \\ &= P_n(x)P_n(y) + \frac{k_{n-1}\,h_n}{k_n\,h_{n-1}} \cdot \frac{P_n(x)P_{n-1}(y) - P_{n-1}(x)P_n(y)}{x-y} \\ &= P_n(x)P_n(y) + h_n \sum_{i=0}^{n-1} \frac{P_i(x)P_i(y)}{h_i} \end{aligned}$$

Division durch h_n liefert dann die Behauptung. □

Bemerkung 3.4.17 (*Rodrigues*-Formel) *Für Orthogonalpolynome gilt zur Angabe der P_n die sog.* Rodrigues[6]*-Formel; d.h. alle P_n erfüllen die folgende explizite Formel*

$$P_n(x) = \frac{1}{e_n \lambda(x)} \frac{d^n}{dx^n}\Big\{\lambda(x)\big(g(x)\big)^n\Big\}, \quad n \in \mathbb{N}_0 \tag{3.38}$$

wobei $g(x)$ ein von n unabhängiges Polynom sei. Die Koeffizienten e_n ergeben sich in einigen wichtigen Fällen aus der Tabelle des nachfolgenden Beispiels.

[4] Christoffel, Elwin Bruno (1829-1900)
[5] Darboux, Jean Gaston (1842-1917)
[6] Rodrigues, Olinde (1794-1851)

Beispiel 3.4.18 (Historisch wichtige Polynome) In der nachfolgenden Tabelle werden einige, v.a. historisch wichtige Polynomterme P_n, ihre Gewichte und Standardisierungen sowie weitere zentrale Eigenschaften aufgelistet.

Wichtige Beispiele von Orthogonalpolynomen								
$P_n(x)$	Name	a	b	$\omega(x)$	Standard.	h_n	e_n	$g(x)$
$P_n(x)$	*Legendre*	-1	1	1	$P_n(1)=1$	$\frac{2}{2n+1}$	$(-2)^n n!$	$1-x^2$
$T_n(x)$	*Tscheby.*	-1	1	$1/\sqrt{1-x^2}$	$T_n(1)=1$	$\binom{\pi/2,\, n\neq 0}{\pi,\, n=0}$	$(-2)^n n!$	$1-x^2$
$L_n(x)$	*Laguerre*[7]	0	∞	e^{-x}	$k_n=(-1)^n/n!$	1	$1/n!$	x
$H_n(x)$	*Hermite*	$-\infty$	∞	e^{-x^2}	$k_n=2^n$	$\sqrt{\pi}\,2^n\,n!$	$(-1)^n$	1
$P_n^{(\alpha,\beta)}(x)$	*Jacobi*	-1	1	$(1-x)^\alpha\cdot$	$P_n^{(\alpha,\beta)}(1)=\binom{n+\alpha}{n}$		$(-2)^n n!$	$1-x^2$
	$\alpha,\beta>-1$			$\cdot(1+x)^\beta$				

Für die *Jacobi*-Polynome gilt

$$h_n(P_n^{(\alpha,\beta)}) = \frac{2^{\alpha-\beta+1}}{2n+\alpha+\beta+1}\,\frac{\Gamma(n+\alpha+1)\Gamma(n+\beta+1)}{n!\,\Gamma(n+\alpha+\beta+1)}\,.$$

Aufgabe 3.4.19 Mit Hilfe der *Rodrigues*-Formel zeige man, dass für die *Legendre*-Polynome gilt

$$\int_{-1}^{1} P_m(x)P_n(x)\,dx = \begin{cases} 0 & \text{falls } m\neq n \\ \frac{2}{2n+1} & \text{falls } m=n\,. \end{cases} \qquad (m,n\in\mathbb{N}_0)$$

Bemerkung 3.4.20 (Eigenschaften der *Jacobi*-Polynome) *Weiterhin gelten für die* Jacobi-*Polynome folgende Eigenschaften:*

(i) $P_n^{(0,0)}(x) = P_n(x)$

(ii) $P_n^{(-1/2,-1/2)}(x) = \binom{n-1/2}{n}T_n(x) = \frac{1}{4^n}\binom{2n}{n}T_n(x)$

(iii) $(1-x^2)\Big(P_n^{(\alpha,\beta)}\Big)'' + \big(\beta-\alpha-(\alpha+\beta+2)x\big)\Big(P_n^{(\alpha,\beta)}\Big)' + n(n+\alpha+\beta+1)P_n^{(\alpha,\beta)} = 0$

Bemerkung 3.4.21 (Spezialfall der Summenformel von *Christoffel-Darboux*) *Man beachte den Spezialfall $y\to x$ in Satz 3.4.16. Es gilt*

$$\begin{aligned}\sum_{i=0}^{n}\frac{P_i^2(x)}{h_i} &= \frac{k_n}{k_{n+1}h_n}\cdot\lim_{y\to x}\frac{P_{n+1}(x)P_n(y)-P_n(x)P_{n+1}(y)}{x-y}\\ &= \frac{k_n}{k_{n+1}h_n}\cdot\lim_{y\to x}\frac{\big(P_{n+1}(x)-P_{n+1}(y)\big)P_n(y)-P_{n+1}(y)\big(P_n(x)-P_n(y)\big)}{x-y}\\ &= \frac{k_n}{k_{n+1}h_n}\Big(P'_{n+1}(x)P_n(x)-P'_n(x)P_{n+1}(x)\Big)\,.\end{aligned}$$

Damit gilt insbesondere für Nullstellen x^* von P_{n+1}

$$P'_{n+1}(x^*)\,P_n(x^*) = \frac{k_{n+1}h_n}{k_n}\sum_{i=0}^{n}\frac{P_i^2(x^*)}{h_i}\,. \qquad (3.39)$$

Wir haben bereits in Abschnitt gesehen, dass der Interpolationsfehler minimierert werden kann, wenn man als Stützstellen die Nullstellen der *Tschebyscheff*-Polynome, d.h. die Nullstellen spezieller Orthogonalpolynome, wählt. Die Nullstellen von Orthogonalpolynomen werden auch der Schlüssel für die *Gauß*-Quadratur sein.

[7] Laguerre, Edmond Nicolas (1834-1886).

Satz 3.4.22 (Einfachheit der Nullstellen von Orthogonalpolynomen) *Es sei $\omega \in C(a,b)$, $\omega(x) > 0$, $x \in (a,b)$ eine positive Gewichtsfunktion, $(\cdot,\cdot)_\omega$ das zugehörige Skalarprodukt. Dann hat ein Orthogonalpolynom $P_k(x)$ von echtem Grad k genau k einfache Nullstellen in (a,b).*

Beweis. Es seien $x_0, \ldots, x_\ell$ die $\ell + 1 \leq k$ verschiedenen Nullstellen $a < x_i < b$, an denen P_k sein Vorzeichen wechselt. Das Polynom

$$Q(x) := (x - x_0) \cdot \cdots \cdot (x - x_\ell)$$

wechselt dann an den gleichen Stellen sein Vorzeichen, sodass die Funktion $\omega(x)P_k(x)Q(x)$ ihr Vorzeichen in (a,b) nicht ändert ($\omega(x)$ ist eine positive Gewichtsfunktion) und daher gilt

$$(Q, P_k)_\omega = \int_a^b \omega(x)Q(x)P_k(x)\,dx \neq 0.$$

Da jedoch P_k auf allen Polynomen aus $\mathbb{P}_{k-1}$ senkrecht und $Q \in P_\ell, \ell < k$, steht, folgt unmittelbar: grad $Q = \ell + 1 \geq k$ und damit die Behauptung. □

Mit der numerischen Berechnung dieser Nullstellen werden wir uns in Abschnitt **??** im Rahmen der Gauß-Quadratur genauer beschäftigen.

Zum Abschluss dieses Abschnitts über Orthogonalpolynom betrachten wir nun noch drei Beispiele für Orthogonalpolynom genauer.

3.4.1.1 *Tschebyscheff*-Polynome

Die *Tschebyscheff*-Polynome T_n, welche wir bereits kennengelernt haben, sind orthogonal bezüglich des Skalarprodukts

$$(f,g)_\omega := \int_{-1}^{1} \frac{f(y)g(y)}{\sqrt{1-y^2}} dy$$

und werden standardisiert durch $T_n(1) = 1$. Durch Einsetzen dieser Eigenschaft in die Formeln von Satz 3.4.12 gewinnt man für $y \in \mathbb{R}$ die **Drei-Term-Rekursion** (2.18), d.h.,

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

Die *Tschebyscheff*-Polynome besitzen weiterhin die folgenden **Eigenschaften**:

(i) Sie haben stets ganzzahlige Koeffizienten.

(ii) Der höchste Koeffizient von T_n ist $a_n = 2^{n-1}$.

(iii) T_n ist stets eine gerade Funktion, falls n gerade, und eine ungerade, falls n ungerade ist.

(iv) $T_n(1) = 1$, $T_n(-1) = (-1)^n$.

(v) $|T_n(y)| \leq 1$ für $y \in [-1,1]$.

(vi) Die Nullstellen von $T_n(y)$ sind

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

(vii)

$$T_k(y) = \begin{cases} \cos(k \cdot \arccos(y)), & |y| \leq 1\ , \\ \cosh(k \cdot \operatorname{Arccosh}(y)), & y > 1\ , \\ (-1)^k \cosh(k \cdot \operatorname{Arccosh}(-y)), & y < -1\ . \end{cases}$$

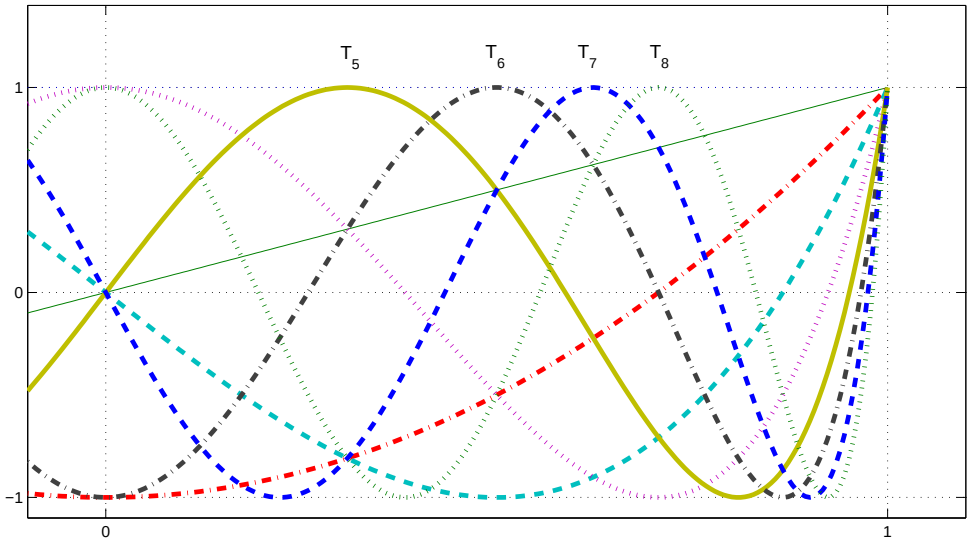


Abb. 3.3: Tschebyscheff-Polynome T_n, $n = 0, \ldots, 8$.

$(viii)$ Die *Tschebyscheff*-Polynome besitzen die globale Darstellung

$$T_k(y) = \frac{1}{2}((y + \sqrt{y^2-1})^k + (y - \sqrt{y^2-1})^k), \quad \text{wobei } y \in \mathbb{R}.$$

(ix) $|T_n(y)|$ nimmt seinen maximalen Wert im Intervall $[-1, 1]$ an den sogenannten *Tschebyscheff*-Abszissen $\overline{y}_k = \cos(\frac{k\pi}{n})$ für $k = 0, \ldots, n$ an, d.h.

$$|T_n(y)| = 1 \quad \Leftrightarrow \quad y = \overline{y}_k = \cos\left(\frac{k\pi}{n}\right) \quad \text{mit } k = 0, \ldots, n.$$

Abschließend möchten wir für $n = 0, \ldots, 5$ die T_n explizit angeben und diese sowie T_6, T_7 und T_8 auch anhand der Grafik 3.3 veranschaulichen:

$$\begin{aligned} T_0(y) &= 1, & T_3(y) &= 4y^3 - 3y, \\ T_1(y) &= y, & T_4(y) &= 8y^4 - 8y^2 + 1, \\ T_2(y) &= 2y^2 - 1, & T_5(y) &= 16y^5 - 20y^3 + 5y. \end{aligned}$$

3.4.1.2 *Legendre*-Polynome

Die *Legendre*-Polynome $P_n \in \mathbb{P}_n$, $n = 0, 1, 2, \ldots$ sind orthogonal bezüglich des L^2-Skalarprodukts auf $[-1, 1]$, d.h.

$$(f, g)_\omega := \int_{-1}^{1} f(y)g(y)\, dy,$$

und sind durch $P_n(1) = 1$ standardisiert.

Die *Legendre*-Polynome besitzen folgende **Eigenschaften**:

(i) Sie erfüllen die **Drei-Term-Rekursion**

$$P_0(y) = 1, \quad P_1(y) = x, \quad nP_n(y) = (2n-1)yP_{n-1}(y) - (n-1)P_{n-2}(y), \quad n \geq 2$$

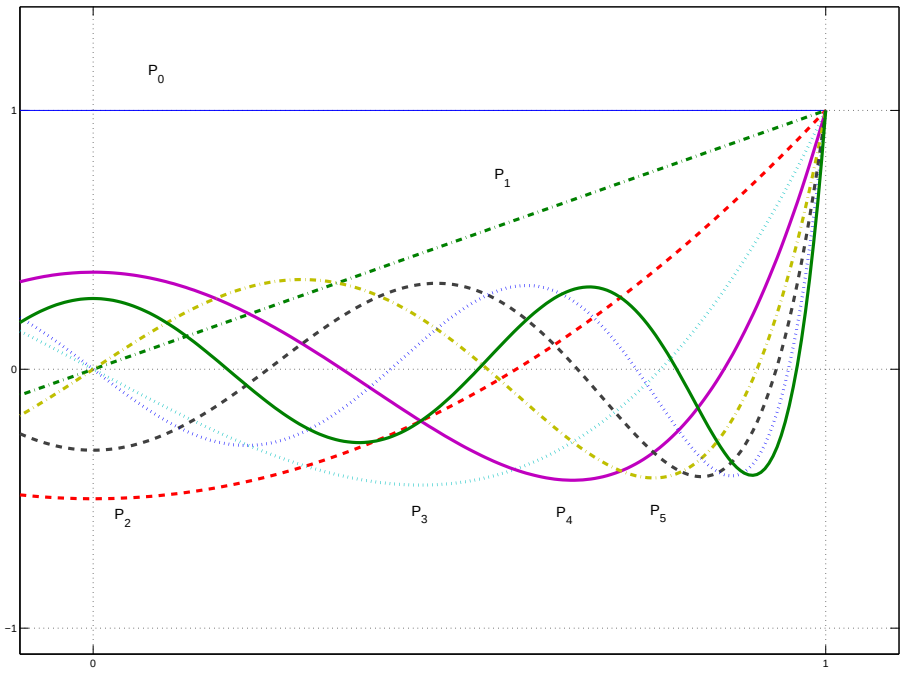


Abb. 3.4: *Legendre*-Polynome P_n, $n = 0, \ldots, 8$.

(ii) $P_n(1) = 1, P_n(-1) = (-1)^n, \quad n = 0, 1, 2, \ldots$

(iii)

$$\int_{-1}^{1} P_n^2(y)\, dy = \frac{2}{2n+1}, \quad n = 0, 1, 2, \ldots \tag{3.40}$$

(iv) Für Ableitung und Stammfunktion gilt

$$P_n'(y) = \frac{n(n+1)}{2n+1} \frac{P_{n+1}(y) - P_{n-1}(y)}{y^2 - 1}, \quad n \geq 1 \tag{3.41}$$

bzw.

$$\int_{-1}^{y} P_n(\xi)\, d\xi = \frac{1}{2n+1} \Big(P_{n+1}(y) - P_{n-1}(y) \Big), \quad n \geq 1\,. \tag{3.42}$$

Im Folgenden sind $P_0, \ldots, P_5$ explizit angegeben und in Grafik 3.4 zusammen mit P_6, P_7 und P_8 aufgezeichnet:

$$\begin{aligned} P_0(y) &= 1, & P_3(y) &= \frac{1}{2}(5y^3 - 3y), \\ P_1(y) &= y, & P_4(y) &= \frac{1}{8}(35y^4 - 30y^2 + 3), \\ P_2(y) &= \frac{1}{2}(3y^2 - 1), & P_5(y) &= \frac{1}{8}(63y^5 - 70y^3 + 15y). \end{aligned}$$

Aufgabe 3.4.23 (i) Man zeige für $m \geq 1$

$$\int_{-1}^{1} (1 - y^2) P_m'(y) P_j(y)\, dy = \begin{cases} \frac{2m(m+1)}{(2m-1)(2m+1)} & \text{falls } j = m-1, \\ -\frac{2m(m+1)}{(2m+1)(2m+3)} & \text{falls } j = m+1, \\ 0 & \text{sonst.} \end{cases}$$

(ii) Man beweise (3.41) mit Hilfe von i).

(iii) Man zeige für $i, j \in \mathbb{N}_0$

$$\int_{-1}^{1} P_i(y) P_j'(y)\, dy = \begin{cases} 2 & \text{falls } i < j \text{ und } j+i \text{ ungerade,} \\ 0 & \text{sonst.} \end{cases}$$

(iv) Man beweise (3.42) mit Hilfe von iii).

(v) Man zeige für $n \in \mathbb{N}$

$$\int_{-1}^{y} P_n(\xi)\, d\xi = \frac{1}{n(n+1)} \left(x^2 - 1\right) P_n'(y)\,.$$

3.4.1.3 *Jacobi*-Polynome

Die *Jacobi*-Polynome $P_n^{(\alpha,\beta)}$ sind orthogonal bezüglich des durch die positive Gewichtsfunktion $w(y) = (1-y)^\alpha (1+y)^\beta, \alpha > -1, \beta > -1$ auf dem Intervall $(-1, 1)$ induzierten Skalarprodukts

$$(f, g)_\omega := \int_{-1}^{1} (1-y)^\alpha (1+y)^\beta f(y) g(y)\, dy.$$

Somit sind *Legendre*- und *Tschebyscheff*-Polynome spezielle *Jacobi*-Polynome (vgl. hierzu auch Bemerkung 3.4.20), für die wir im Folgenden eine Aussage über die Verteilung der Nullstellen wiedergeben:

Satz 3.4.24 ([Sz]) *Seien $|\alpha|, |\beta| \leq 1/2$ und $x_k = \cos\theta_k$ die Nullstellen von $P_n^{(\alpha,\beta)}(x)$ in absteigender Folge, d.h.*

$$1 > x_0 > x_1 > \ldots > x_{n-1} > -1;\ 0 < \theta_0 < \theta_1 < \ldots < \theta_{n-1} < \pi\,.$$

Dann gilt

$$\frac{k+1+(\alpha+\beta-1)/2}{n+(\alpha+\beta+1)/2}\pi < \theta_k < \frac{k+1}{n+(\alpha+\beta+1)/2}\pi,\ \ k = 0, 1, 2, \ldots$$

und im Falle $\alpha = \beta$ gilt

$$\theta_k \geq \frac{k+1+\alpha/2-1/4}{n+\alpha+1/2}\pi,\ \ k = 0, 1, 2, \ldots, [(n-1)/2],$$

wobei die Gleichheit für $\alpha = \beta = -1/2$ oder $\alpha = \beta = 1/2$ gilt.

Beweis. Wir verzichten an dieser Stelle auf die Wiedergabe eines Beweises dieser Aussage und verweisen auf [Sz]. □

3.4.2 Konstruktion von Gauß-Quadraturen

Nun zurück zur Gauß-Quadratur, d.h. zur Konstruktion einer Quadraturformel mit maximaler Fehlerordnung $2n+2$ zu $n+1$ Stützstellen. Im Beweis des Satzes 3.4.1 haben wir gesehen, dass das Polynom

$$P(x) = \prod_{i=0}^{n} (x - x_i^{(n)})^2 = \prod_{i=0}^{n} (x - x_i^{(n)})(x - x_i^{(n)}) \in \mathbb{P}_{2n+2}$$

durch eine Quadraturformel $\hat{I}_n$ mit den Knoten $x_i^{(n)}$ *nicht* exakt integriert wird. Die Frage stellt sich, was man über die Knoten sagen kann, wenn man weiß, dass $\hat{I}_n$ maximalen Exaktheitsgrad $2n+1$ hat?

Lemma 3.4.25 (Charakterisierung via Orthogonalität) *Ist $\hat{I}_n$ der Form*

$$\hat{I}_n(f) = (b-a)\sum_{k=0}^{n} \lambda_k^{(n)} f(x_k^{(n)})$$

exakt vom Grad $2n+1$, dann gilt für die Polynome

$$P_{k+1}(x) := \prod_{i=0}^{k}(x - x_i^{(k)}) \in \mathbb{P}_{k+1}, \quad k = 0, \ldots, n \tag{3.43}$$

(die Stützstellen $x_i^{(k)}$ sind also die Nullstellen von P_{k+1}) die Beziehung

$$(P_j, P_{n+1})_\omega = \int_a^b P_j(x)P_{n+1}(x)\omega(x)\,dx = 0 \quad \textit{für alle } j = 0, \ldots, n.$$

Beweis. Wegen $P_k \in \mathbb{P}_k$ gilt $P_j P_{n+1} \in \mathbb{P}_{2n+1}$ für alle $0 \le j \le n$, also

$$\begin{aligned}\int_a^b P_j(x)P_{n+1}(x)\omega(x)dx &= I(P_jP_{n+1}) = \hat{I}_n(P_jP_{n+1}) \\ &= (b-a)\sum_{i=0}^{n} \lambda_i^{(n)} P_j(x_i^{(n)}) \underbrace{P_{n+1}(x_i^{(n)})}_{=0} = 0.\end{aligned}$$

□

Bemerkung 3.4.26 *Nullstellen von orthogonalen Polynomen scheinen eine gute Wahl für die gesuchten optimalen Knoten zu sein!*

Damit sind auch die Gewichte bestimmt: falls $\hat{I}_n$ exakt von der Ordnung n ist, muss dies insbesondere für die *Lagrange*-Polynome $L_i^{(n)}$ bzgl. $x_k^{(n)}$ gelten.

$$\begin{aligned}0 &= I(L_i^{(n)}) - \hat{I}_n(L_i^{(n)}) \\ &= \int_a^b L_i^{(n)}(x)\omega(x)\,dx - (b-a)\sum_{j=0}^{n} \lambda_j^{(n)} \underbrace{L_i^{(n)}(x_j^{(n)})}_{=\delta_{ij}} \\ &= \int_a^b L_i^{(n)}(x)\omega(x)\,dx - (b-a)\lambda_i^{(n)},\end{aligned}$$

also

$$\lambda_i^{(n)} = \frac{1}{(b-a)}\int_a^b L_i^{(n)}(x)\omega(x)\,dx. \tag{3.44}$$

Damit gilt (erstaunlicherweise):

Satz 3.4.27 *Seien $x_k^{(n)}$, $k = 0, \ldots, n$ die Nullstellen von P_{n+1}, dann gilt für $\hat{I}_n$ mit den Gewichten (3.44) folgende Aussage: $\hat{I}_n$ ist exakt auf $\mathbb{P}_n$ genau dann, wenn $\hat{I}_n$ exakt auf $\mathbb{P}_{2n+1}$ ist.*

Beweis. „⇐“ ist trivial. Sei also umgekehrt $\hat{I}_n$ exakt auf $\mathbb{P}_n$ und $P \in \mathbb{P}_{2n+1}$. Mit dem Euklidischen Algorithmus folgt, dass Polynome $Q, R \in \mathbb{P}_n$ existieren mit

$$P = QP_{n+1} + R. \tag{3.45}$$

Die orthogonalen Polynome $(P_k)_{k\leq n}$ bilden eine Basis für $\mathbb{P}_n$ (aufgrund der linearen Unabhängigkeit), also folgt

$$(Q, P_{n+1})_\omega = 0$$

für alle $Q \in \mathcal{P}_n$ und damit (wegen $R \in P_n$)

$$I(P) = I(QP_{n+1}) + I(R) = \underbrace{(Q, P_{n+1})_\omega}_{=0} + I(R) = \hat{I}_n(R)$$

und andererseits

$$\begin{aligned} \hat{I}_n(R) &= (b-a)\sum_{i=0}^{n} \lambda_i^{(n)} R(x_i^{(n)}) = (b-a)\sum_{i=0}^{n} \lambda_i^{(n)} \left\{ Q(x_i^{(n)}) \underbrace{P_{n+1}(x_i^{(n)})}_{=0} + R(x_i^{(n)}) \right\} \\ &= \hat{I}_n(P), \end{aligned}$$

also $\hat{I}_n(P) = I(P)$ für alle $P \in \mathbb{P}_{2n+1}$. □

Wir fassen dies Überlegungen in dem folgenden Satz zusammen:

Satz 3.4.28 (Gauß-Christoffel-Quadratur) *Es sei $\omega \in C(a,b)$ eine positive Gewichtsfunktion auf (a,b) und $(P_k)_{k\in\mathbb{N}_0}$ eine Folge von Orthogonalpolynomen bzgl. des Skalarprodukts $(\cdot,\cdot)_\omega$. Des Weiteren sei $n \in \mathbb{N}$ und $x_0 < \cdots < x_n$ die Nullstellen von $P_{n+1}(x)$. Dann existieren (von n abhängige) eindeutig bestimmte Gewichte $\lambda_0, \cdots, \lambda_n \in \mathbb{R}_+$, so dass gilt:*

$$\int_a^b \omega(x)P(x)\,dx = \lambda_0\, q(x_1) + \ldots + \lambda_n\, P(x_n) \quad \text{für alle } P \in \mathbb{P}_{2n+1}\,. \tag{3.46}$$

Beweis. Es sei P_{n+1} das bis auf Vielfachheit eindeutig bestimmte Orthogonalpolynom bzgl. $(\cdot,\cdot)_\omega$ vom echten Grad $n+1$. Dann existiert zu jedem $P \in \mathbb{P}_{2n+1}$ die eindeutige Darstellung $P = QP_{n+1} + R$ mit $Q, R \in \mathbb{P}_n$. Das Ausnutzen der Orthogonalität von $(P_{n+1}, x^k)_\omega = 0$ für $k = 0, \ldots, n$ liefert

$$\int_a^b \omega(x)P(x)\,dx = \int_a^b \omega(x)Q(x)P_{n+1}(x)\,dx + \int_a^b \omega(x)R(x)\,dx = \int_a^b \omega(x)R(x)\,dx. \tag{3.47}$$

Seien $L_0^{(n)}(x), \ldots, L_n^{(n)}(x) \in \mathbb{P}_n$ die *Lagrange*-Polynome zu $x_0, \ldots, x_n$. Man beachte die Eigenschaft $L_k(x_j) = \delta_{jk}$, wobei δ_{jk} das Kronecker-Symbol ist. Es gilt dann folgende Darstellung

$$\psi(x) = \sum_{i=0}^{n} \psi(x_i) L_i^{(n)}(x)$$

und somit für (3.47)

$$\int_a^b \omega(x)P(x)\,dx = \int_a^b \omega(x) \left(\sum_{i=0}^{n} R(x_i) L_i^{(n)}(x) \right) dx = \sum_{i=0}^{n} R(x_i) \int_a^b \omega(x) L_i^{(n)}(x)\,dx.$$

Dies ist aber genau die Darstellung (3.46) mit $\lambda_i = \int_a^b \omega(x) L_i^{(n)}(x)dx$, da an den Nullstellen von P_{n+1} gilt $P(x_i) = Q(x_i)\,P_n(x_i) + R(x_i) = R(x_i)$.

Beweisen wir nun die **Eindeutigkeit** der λ_i. Dazu seien $\lambda_0, \dots, \lambda_n$ und $\widetilde{\lambda_0}, \dots, \widetilde{\lambda_n}$ zwei verschiedene Wahlen der Koeffizienten in (3.46). Durch Differenzbildung erhalten wir also

$$0 = \sum_{0=1}^{n} R(x_i)(\lambda_i - \widetilde{\lambda_i}),$$

setzen wir nacheinander $R(x) = L_k^{(n)}(x)$, $k = 0, \dots, n$, so folgt sofort $\lambda_k = \widetilde{\lambda_k}$ für alle $k \in \{0, 1 \dots, n\}$.

Zuletzt müssen wir noch die **Positivität** der λ_i nachweisen. Setzen wir dazu $P(x) = (L_k^{(n)}(x))^2$, dann folgt mit $\lambda(x) > 0$, $x \in (a, b)$ und (3.46)

$$0 < \int_a^b \omega(x) \left(L_k\right)^{(n)}(x)\Big)^2 \, dx = \sum_{0=1}^{n} \lambda_i \, \left(L_k^{(n)}(x_i)\right)^2 = \lambda_k$$

und damit die Behauptung. □

Bemerkung 3.4.29 (*Gauß-Christoffel*-Quadratur) *(a) Für $\omega = 1$ spricht man von* **Gauß -*Quadratur***, *allgmein von* **Gauß-Christoffel-*Quadratur***.*

(b) Nachteile sind einerseits, dass die Berechnung der Gewichte und Knoten vom Integrationsintervall ab hängt, und andererseits, dass diese für jedes n neu berechnet werden müssen. Eine nachträgliche Erhöhung der Genauigkeit durch Aufdatierung ist so noch nicht möglich, es muss alles neu berechnet werden!

(c) Die Berechnung der Gewichte und Knoten für beliebige ω wird mit der Drei-Term-Rekursion gemacht. Dies behandeln wir im nächsten Abschnitt genauer.

*(d) Integrale auf mehrdimensionalen Gebieten („*Kubatur*")*

$$[a_1, b_1] \times \cdots \times [a_d, b_d]$$

kann man mit Tensorprodukt-Quadratur-Formeln

$$\hat{I}_{n_1}(x_1) \cdot \cdots \cdot \hat{I}_{n_d}(x_d)$$

*berechnen. Die Komplexität steigt jedoch stark mit der Raumdimension d („*Fluch der Dimensionen*"). Mögliche Auswege sind Monte-Carlo, Quasi-Monte-Carlo, Dünngitter (siehe Numerical Finance).*

Nun zur Darstellung des Fehlers. Analog zu den interpolatorischen Quadraturformeln kann man vom Exaktheitsgrad auf den Approximationsfehler schließen.

Satz 3.4.30 (Integrationsfehler der Gauß-Quadratur) *Für $f \in C^{2n+2}$ gilt*

$$E_n(f) = I(f) - \hat{I}_n(f) = \frac{f^{2n+2}(\xi)}{(2n+2)!} \|P_{n+1}\|_\omega^2 \tag{3.48}$$

mit einem $\xi \in (a, b)$.

Beweis. Betrachte den Hermite-Interpolanten $P \in \mathbb{P}_{2n+1}$ an f zu den jeweils doppelten Knoten $x_k^{(n)}$, $k = 0, \dots, n$. Dann folgt

$$f(x) - P(f|x_0^{(n)}, x_0^{(n)}, \dots, x_n^{(n)}, x_n^{(n)})(x) = \underbrace{(x - x_0^{(n)})^2 \dots (x - x_n^{(n)})^2}_{=P_{n+1}(x)^2} \frac{f^{2n+2}(\xi)}{(2n+2)!},$$

also

$$I(f) = \underbrace{I(P)}_{=\hat{I}_n(P)} + \frac{f^{2n+2}(\xi)}{(2n+2)!} I(P_{n+1}^2) = \underbrace{(b-a)\sum_{i=0}^{n} \lambda_i^{(n)} \underbrace{P(x_i^{(n)})}_{=f(x_i^{(n)})}}_{=\hat{I}_n(f)} + \frac{f^{2n+2}(\xi)}{(2n+2)!} \underbrace{(P_{n+1}, P_{n+1})_\omega}_{=\|P_{n+1}\|_\omega^2}.$$

□

3.4.3 Berechnung der Knoten und Gewichte

Aus der Summenformel von *Christoffel-Darboux* bzw. dem Spezialfall, welcher in Bemerkung 3.4.21 behandelt wird, ergibt sich die folgende Darstellung der Gewichte λ_k zu den Knoten $x_k, k = 0, 1, \ldots, n$.

Satz 3.4.31 (Darstellung der Gewichte) *Für die Gewichte λ_k, $k = 0, 1, \ldots, n$ in Satz 3.4.28 gelten die Darstellungen*

$$\lambda_k = \frac{k_{n+1}\, h_n}{P'_{n+1}(x_k)\, P_n(x_k)\, k_n} = \left(\sum_{i=0}^{n} \frac{P_i^2(x_k)}{h_i} \right)^{-1}, \quad k = 0, 1, \ldots, n\,,$$

wobei x_k die Nullstellen von P_{n+1} seien.

Beweis. Es sei x_k, $k \in \{0, 1, \ldots, n\}$, eine Nullstelle von P_{n+1}. Dann gilt

$$\lim_{x \to x_k} \frac{P_{n+1}(x)}{x - x_k} = \lim_{x \to x_k} \frac{P_{n+1}(x) - P_{n+1}(x_k)}{x - x_k} = P'_{n+1}(x_k)\,.$$

Wir definieren

$$Q(x) = \begin{cases} P_{n+1}(x)\, P_n(x)/(x - x_k) & \text{falls } x \in \mathbb{R} \setminus \{x_k\} \\ P'_{n+1}(x_k)\, P_n(x_k) & \text{falls } x = x_k \end{cases}.$$

Da $Q \in \mathbb{P}_{2n}$, gilt

$$\int_a^b \omega(x) Q(x)\, dx = \sum_{j=0}^{n} \lambda_j\, Q(x_j) = \lambda_k\, Q(x_k) = \lambda_k\, P'_{n+1}(x_k)\, P_n(x_k)\,. \tag{3.49}$$

Nutzen wir die Darstellung $P_{n+1}(x) = \frac{k_{n+1}}{k_n}(x - x_k)P_n(x) + (x - x_k)R(x)$ mit einem $R \in \mathbb{P}_{n-1}$, so ergibt sich

$$\begin{aligned} \int_a^b \omega(x) Q(x)\, dx &= \int_a^b \omega(x) \left(\frac{k_{n+1}}{k_n} P_n(x) + R(x) \right) P_n(x)\, dx \\ &= \frac{k_{n+1}}{k_n} \int_a^b \omega(x) P_n^2(x)\, dx = \frac{k_{n+1}\, h_n}{k_n}\,. \end{aligned} \tag{3.50}$$

Aus (3.49) und (3.50) erhalten wir die erste und daraus mit (3.39) die zweite Darstellung . □

Zur numerischen Berechnung der Knoten und Gewichte greift man auf die Eigenschaften der Orthogonalpolynome zurück: Aus Abschnitt wissen wir, dass die Orthogonalpolynome zu einem gewichteten Skalarprodukt durch eine Drei-Term-Rekursion berechnet werden können. Mit Hilfe dieser lässt sich ein lineares Gleichungssystem formulieren mit einer Tridiagonalmatrix. Die Nullstellen der Orthogonalpolynome sind dann die Eigenwerte dieser Matrix.

Wie in Satz 3.4.15 bewiesen wurde, erfüllen die Orthogonalpolynome folgende Drei-Term-Rekursion mit den Startwerten $P_{-1} = 0$, $P_0 \in \mathbb{P}_0$:

$$P_n(x) = (a_n + b_n x)P_{n-1} + c_n P_{n-2}, \quad n = 1, 2, \ldots.$$

Schematisch dargestellt bedeutet dies:

$$\begin{aligned}
P_1 &= (a_1 + b_1 x)P_0 && \Leftrightarrow && -\frac{a_1}{b_1} \cdot P_0 + \frac{1}{b_1} \cdot P_1 = xP_0 \\
P_2 &= (a_2 + b_2 x)P_1 + c_2 P_0 && \Leftrightarrow && -\frac{c_2}{b_2} \cdot P_0 - \frac{a_2}{b_2} \cdot P_1 + \frac{1}{b_2} \cdot P_2 = xP_1 \\
&\vdots && && \vdots \\
P_{n+1} &= (a_{n+1} + b_{n+1} x)P_n + c_{n+1} P_{n-1} && \Leftrightarrow && -\frac{c_{n+1}}{b_{n+1}} \cdot P_{n-1} - \frac{a_{n+1}}{b_{n+1}} \cdot P_n = xP_n - \frac{1}{b_{n+1}} \cdot P_{n+1}
\end{aligned}$$

Aus dem Schema ist aber ersichtlich, dass wir obige Rekursion auch als folgendes lineares Gleichungssystem interpretieren können, d.h.

$$\underbrace{\begin{pmatrix}
-\frac{a_1}{b_1} & \frac{1}{b_1} & 0 & \cdots & \cdots & 0 \\
-\frac{c_2}{b_2} & -\frac{a_2}{b_2} & \frac{1}{b_2} & & & \vdots \\
0 & -\frac{c_3}{b_3} & -\frac{a_3}{b_3} & \frac{1}{b_3} & & \vdots \\
\vdots & & \ddots & \ddots & \ddots & 0 \\
\vdots & & & \ddots & \ddots & \frac{1}{b_n} \\
0 & \cdots & & 0 & -\frac{c_{n+1}}{b_{n+1}} & -\frac{a_{n+1}}{b_{n+1}}
\end{pmatrix}}_{=: A} \cdot \begin{pmatrix} P_0(x) \\ \vdots \\ \\ \\ \vdots \\ P_n(x) \end{pmatrix} = x \cdot \underbrace{\begin{pmatrix} P_0(x) \\ \vdots \\ \\ \\ \vdots \\ P_n(x) \end{pmatrix}}_{=: p} + \underbrace{\begin{pmatrix} 0 \\ \vdots \\ \\ \\ \vdots \\ 0 \\ -\frac{P_{n+1}(x)}{b_{n+1}} \end{pmatrix}}_{=: r}$$

In Matrixschreibweise also

$$Ap = xp + r. \tag{3.51}$$

Aus dieser Darstellung ergibt sich das folgende bemerkenswerte Resultat:

Satz 3.4.32 (Nullstellen von Orthogonalpolynomen als Lösung eines Eigenwertproblems)
Die Nullstellen von P_{n+1} sind genau die Eigenwerte von A, d.h. die Lösungen der Gleichung $Ap = xp$ für $p \neq 0$. Die Eigenvektoren zu Eigenwerten x_k, $k = 0, \ldots, n$ lauten bis auf ein Vielfaches $(P_0(x_k), \ldots, P_n(x_k))^T$.

Beweis. Der Beweis ist nicht sehr schwer und bleibt dem Leser als Aufgabe überlassen. □

Da P_{n+1} ausschießlich reelle Nullstellen besitzt, liegt die Idee nahe, dass sich A in eine symmetrische Matrix mittels einer Ähnlichkeitstransformation transformieren lässt. Die einfachste Möglichkeit wäre eine Skalierung mit einer Diagonalmatrix.

Da die Orthogonalität die Orthogonalpolynome nur bis auf ein Vielfaches eindeutig bestimmt, steht es uns frei, $b_k > 0, k = 1, 2, 3, \ldots$ zu wählen, zumal die Nullstellen der Orthogonalpolynome davon nicht betroffen sind. Wegen $b_k > 0$ folgt somit aus $h_k > 0$ und (3.32) $c_k < 0$, $k = 2, 3, \ldots$.

Satz 3.4.33 *Es seien $a_j \in \mathbb{R}$, $b_j > 0$,$j = 1, \ldots, n+1$, $c_j < 0$, $j = 2, \ldots, n+1$ und*

$$A := \begin{pmatrix} -\frac{a_1}{b_1} & \frac{1}{b_1} & 0 & \cdots & \cdots & 0 \\ -\frac{c_2}{b_2} & -\frac{a_2}{b_2} & \frac{1}{b_2} & & & \vdots \\ 0 & -\frac{c_3}{b_3} & -\frac{a_3}{b_3} & \frac{1}{b_3} & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & 0 \\ \vdots & & & \ddots & \ddots & \frac{1}{b_n} \\ 0 & \cdots & & 0 & -\frac{c_{n+1}}{b_{n+1}} & -\frac{a_{n+1}}{b_{n+1}} \end{pmatrix} \in \mathbb{R}^{n \times n}. \tag{3.52}$$

Dann existiert eine Diagonalmatrix $D \in \mathbb{R}^{(n+1)\times(n+1)}$, so dass $\widehat{A} := (\widehat{a}_{ij})_{i,j=1}^{n+1} = DAD^{-1}$ symmetrisch ist, und es gilt

$$\widehat{a}_{ii} = -\frac{a_i}{b_i}, \qquad \widehat{a}_{i,i-1} = \widehat{a}_{i-1,i} = \frac{|c_i|^{1/2}}{b_i^{1/2} b_{i-1}^{1/2}} \tag{3.53}$$

und für $x = (x_0, \ldots, x_n)^T \in \mathbb{R}^{n+1}$ ergibt sich

$$Dx = \left(x_0 \sqrt{b_1}, x_1 \frac{\sqrt{b_2}}{\sqrt{|c_2|}}, \ldots, x_n \frac{\sqrt{b_{n+1}}}{\sqrt{|c_2 \cdot \ldots \cdot c_{n+1}|}} \right)^T.$$

Beweis. Allgemein gilt für eine Matrix $M = (m_{ij}) \in \mathbb{R}^{(n+1)\times(n+1)}$ und eine Diagonalmatrix $D = \text{diag}(d_1, \ldots, d_{n+1}) \in \mathbb{R}^{(n+1)\times(n+1)}$, dass die Einträge von

$$\widehat{M} = (\widehat{m}_{ij}) := DMD^{-1} \quad \text{die Form } \widehat{m}_{ij} = \frac{d_i}{d_j} m_{ij} \text{ haben.}$$

Damit nun $\widehat{A}$ symmetrisch ist, muss $\widehat{a}_{i,i-1} = \widehat{a}_{i-1,i}$ gelten und somit

$$-\frac{c_i}{b_i}\frac{d_i}{d_{i-1}} = \frac{1}{b_{i-1}}\frac{d_{i-1}}{d_i}, \text{ d.h. } d_i^2 = d_{i-1}^2 \cdot \frac{b_i}{b_{i-1}|c_i|}, \quad i = 2, 3, \ldots, n+1. \tag{3.54}$$

Dies erreichen wir nun z.B., indem wir $d_1 = \sqrt{b_1}$ setzen. Dann folgt mit (3.54)

$$d_2^2 = d_1^2 \frac{b_2}{b_1|c_2|} = \frac{b_2}{|c_2|}, \quad d_3^2 = d_2^2 \frac{b_3}{b_2|c_3|} = \frac{b_2}{|c_2|}\frac{b_3}{b_2|c_3|} = \frac{b_3}{|c_2 c_3|}, \quad d_4^2 = d_3^2 \frac{b_4}{b_3|c_4|} = \frac{b_4}{|c_2 c_3 c_4|}.$$

Allgemein folgt aus

$$d_{k-1}^2 = \frac{b_{k-1}}{|c_2 \cdot c_3 \cdot \ldots \cdot c_{k-1}|}$$

die Darstellung

$$d_k^2 = d_{k-1}^2 \frac{b_k}{b_{k-1}|c_k|} = \frac{b_k}{|c_2 \cdot \ldots \cdot c_k|}.$$

Bei dieser Wahl von D gilt für die Diagonaleinträge $\widehat{a}_{ii}$, $i = 1, \ldots, n+1$, der symmetrischen Tridiagonalmatrix $\widehat{A}$

$$\widehat{a}_{ii} = \frac{d_i}{d_i} a_{ii} = -\frac{a_i}{b_i},$$

und für die Nebendiagonaleinträge

$$\widehat{a}_{i,i-1} = d_i \cdot a_{i,i-1} \cdot d_{i-1}^{-1} = \cdot \frac{b_i^{1/2}}{|c_2 \cdot \ldots \cdot c_i|^{1/2}} \cdot \frac{|c_i|}{b_i} \cdot \frac{|c_2 \cdot \ldots \cdot c_{i-1}|^{1/2}}{b_{i-1}^{1/2}} = \frac{|c_i|^{1/2}}{b_i^{1/2} b_{i-1}^{1/2}} = \widehat{a}_{i-1,i}.$$

Für das Matrix-Vektorprodukt von D mit einem Vektor $x \in \mathbb{R}^{n+1}$ gilt

$$\begin{aligned} Dx &= \text{diag}(d_1, \ldots, d_{n+1})x = \big(d_1\, x_0, d_2\, x_1, \ldots, d_{n+1}\, x_n\big)^T \\ &= \left(x_0\sqrt{b_1}, x_1\frac{\sqrt{b_2}}{\sqrt{|c_2|}}, \ldots, x_{n-1}\frac{\sqrt{b_n}}{\sqrt{|c_2 \cdot \ldots \cdot c_{n+1}|}}\right)^T. \end{aligned}$$

□

Wie wir später sehen werden, lassen sich symmetrische Eigenwertprobleme numerisch besser behandeln. Anstatt also $Au = \lambda\, u$ zu analysieren, betrachten wir

$$DAD^{-1}Du = \lambda\, Du$$

d.h.

$$\widehat{A}\widehat{u} = \lambda\widehat{u}$$

mit $\widehat{A} := DAD^{-1}$ und $\widehat{u} := Du$. Unter dieser Ähnlichkeitstransformation bleiben also die Eigenwerte unverändert und für die zugehörigen Eigenvektoren gilt, dass Eigenvektoren u in Du übergehen. Unter der im Beweis von Satz 3.4.33 verwendeten Ähnlichkeitstransformation lauten somit die Eigenvektoren zu $\widehat{A}\widehat{u} = \lambda\widehat{u}$ zum Eigenwert x_k, $k = 0, 1, \ldots, n$, bis auf eine nichtverschwindende Konstante $c \in \mathbb{R}$

$$\widehat{u}_k = c\left(P_0\sqrt{b_1}, \frac{P_1(x_k)\sqrt{b_2}}{\sqrt{|c_2|}}, \ldots, \frac{P_n(x_k)\sqrt{b_{n+1}}}{\sqrt{|c_2 \cdot \ldots \cdot c_{n+1}|}}\right)^T.$$

Der nachfolgende Satz liefert nun, dass sich die Gewichte aus den Quadraten der ersten Komponenten der normierten Eigenvektoren ergeben.

Satz 3.4.34 (Numerische Berechnung der Gewichte) *Es gelten die Voraussetzungen wie in Satz 3.4.33 und $\widehat{A}$ sei durch* (3.52) *definiert. Des Weiteren sei $\widehat{\mathcal{U}}$ die Matrix, die spaltenweise die Eigenvektoren $\widehat{u}_k = (\widehat{u}_{k1}, \ldots, \widehat{u}_{k(n+1)})^T$ zu den Eigenwerten x_k, $k = 0, 1, \ldots, n$, enthält, d.h.*

$$\widehat{A}\widehat{\mathcal{U}} = \mathit{diag}(x_0, \ldots, x_{n-1})\widehat{\mathcal{U}}\,.$$

Dann sind die Gewichte in Satz 3.4.28 gegeben durch

$$\lambda_k = (1,1)_\omega \frac{\widehat{u}_{k1}^2}{\widehat{u}_k^T\,\widehat{u}_k}, \quad k = 0, 1, \ldots, n\,. \tag{3.55}$$

Beweis. Die Vektoren $\widehat{v}_k$ seien Vielfache der Eigenvektoren $\widehat{u}_k$, $k = 1, \ldots, n+1$, mit der Eigenschaft, dass für den ersten Eintrag jeweils $\widehat{v}_{k1} = p_0\sqrt{b_1}$ gilt. Sei $\mathcal{V} = (\widehat{v}_1|\widehat{v}_2|\cdots|\widehat{v}_{n+1})$ die Matrix, die spaltenweise die Eigenvektoren $\widehat{v}_k$ enthält und $\lambda = (\lambda_0, \lambda_2, \ldots, \lambda_n)^T$ der Vektor der gesuchten Gewichte. Aufgrund der verwendeten Normierung steht in der ersten Zeile jeweils $p_0\sqrt{b_1}$. Nach Satz 3.4.28 integriert die gesuchte Quadraturformel Polynome bis zum Grad $2n+1$ exakt, d.h.

$$\sum_{j=0}^{n} \lambda_j Q(x_j) = \int_a^b \omega(x)Q(x)dx = \begin{cases} (1, P_0)_\omega & \text{für } Q = P_0, \\ 0 & \text{für } Q = P_k,\ 1 \le k \le n+1. \end{cases}$$

Damit folgt

$$\mathcal{V}\,\lambda = \begin{pmatrix} \sum_j \lambda_j P_0\sqrt{b_1} \\ \sum_j \lambda_j P_1(x_j)\sqrt{b_2\,/\,|c_2|} \\ \vdots \\ \sum_j \lambda_j P_n(x_j)\sqrt{b_{n+1}\,/\,|c_2 \cdot \ldots \cdot c_{n+1}|} \end{pmatrix} = P_0\sqrt{b_1}\begin{pmatrix} (1,1)_\omega \\ 0 \\ \vdots \\ 0 \end{pmatrix}. \tag{3.56}$$

Die Nullstellen von P_{n+1} sind einfach nach Satz 3.4.22. Damit folgt aus Satz 3.4.32, dass auch die Eigenwerte von $\widehat{A}$ einfach sind. Für Eigenvektoren $\widehat{v}_k, \widehat{v}_m$ zu verschiedenen Eigenwerten x_k, x_m ($k \neq m$) gilt nun, da $\widehat{A}$ symmetrisch ist

$$x_m\, \widehat{v}_m^T\, \widehat{v}_k = x_m\, \widehat{v}_k^T\, \widehat{v}_m = \widehat{v}_k^T\, \widehat{A}\, \widehat{v}_m = \widehat{v}_m^T\, \widehat{A}^T\, \widehat{v}_k = \widehat{v}_m^T\, \widehat{A}\, \widehat{v}_k = x_k\, \widehat{v}_m^T\, \widehat{v}_k\,.$$

Somit gilt $\widehat{v}_m^T\, \widehat{v}_k = 0$ für $k \neq m$, d.h. Eigenvektoren zu paarweise verschiedenen Eigenwerten stehen senkrecht aufeinander. Für einen beliebigen Eigenvektor $\widehat{v}_k$ ergibt sich unter Ausnutzung von (3.56) und der eben gezeigten Orthogonalität

$$P_0^2 b_1 (1,1)_\omega = \begin{pmatrix} P_0\sqrt{b_1} \\ \frac{P_1(x_k)\sqrt{b_2}}{\sqrt{|c_2|}} \\ \vdots \\ \frac{P_{n-1}(x_k)\sqrt{b_n}}{\sqrt{|c_2 \cdot \ldots \cdot c_{n+1}|}} \end{pmatrix}^T \cdot \begin{pmatrix} P_0\sqrt{b_1}(1,1)_\omega \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \widehat{v}_k^T \mathcal{V}\, \lambda = \widehat{v}_k^T \sum_{i=0}^{n} \lambda_i \widehat{v}_i = \lambda_k\, \widehat{v}_k^T\, \widehat{v}_k\,.$$

Berücksichtigung der vorangegangenen Normierung, d.h. $\widehat{v}_k = p_0\sqrt{b_1}\widehat{u}_k/\widehat{u}_{k1}$, liefert das Ergebnis. □

Bemerkung 3.4.35 *Sind die Momente $m_k := \int_a^b \lambda(x) x^k\, dx$, $k = 0, \ldots, 2n+1$, bekannt, so lassen sich mit Satz 3.4.12 die Koeffizienten α_j, $j = 1, \ldots, n+1$, und γ_j, $j = 2, \ldots, n+1$, in der Drei-Term-Rekursion*

$$P_n(x) = (\alpha_n + x) P_{n-1}(x) + \gamma_n\, P_{n-2}(x)$$

mittels

$$\alpha_n = -\frac{(xP_{n-1}, P_{n-1})_\omega}{(P_{n-1}, P_{n-1})_\omega}, \qquad \gamma_n = -\frac{(P_{n-1}, P_{n-1})_\omega}{(P_{n-2}, P_{n-2})_\omega}$$

bestimmen. Denn gilt $P_k(x) = \sum_{j=0}^k \lambda_j x^j$, $k \geq 0$, so erhält man für

$$(P_k, P_k)_\omega = \sum_{i=0}^{k} \sum_{j=0}^{k} \lambda_i \lambda_j \int_a^b \omega(x) x^{i+j}\, dx = \sum_{i=0}^{k} \sum_{j=0}^{k} \lambda_i \lambda_j \cdot m_{i+j}$$

und

$$(xP_k, P_k)_\omega = \sum_{i=0}^{k} \sum_{j=0}^{k} \lambda_i \lambda_j \cdot m_{i+j+1}\,.$$

Setzt man nun die so berechneten Werte α_j, γ_j in die Tridiagonalmatrix $\widehat{A}$ mit den Einträgen gegeben durch (3.53) *für $a_j = \alpha_j$, $b_j = 1$ und $c_j = \gamma_j$ ein, so sind die Quadraturstellen zur Gewichtsfunktion ω gerade die Eigenwerte von $\widehat{A}$ und die Gewichte lassen sich mittels* (3.55) *aus den zugehörigen Eigenvektoren bestimmen.*

MATLAB-Funktion: GaussWeightsNodes.m

```
function [weights,nodes,kn] = GaussWeightsNodes(momente)
n = (length(momente)+1)/2;
% Orth.polynome und Koeff. mit 3-Term-Rekursion bestimmen
% p_k = ( x - aa_(k-1) ) * p_(k-1) - cc_(k-1) * p_(k-2)
h(1) = momente(1);                % h  = ( p_k , p_k )
hx(1) = momente(2);               % hx = ( x * p_k , p_k )
aa(1) = hx(1)/h(1);
p{1} = 1;
p{2} = [-aa(1);1];
for k = 3:n                       % bestimme p_k mit 3-Term-Rekurs.
  q = mult(p{k-1},p{k-1});        % berechne q = p_(k-1) * p_(k-1)
  h(k-1) = integ(q,momente);      % berechne int_aˆb omega*q dx
  hx(k-1) = integ([0;q],momente);% berechne int_aˆb omega*x*q dx
  aa(k-1,1) = hx(k-1)/h(k-1);     % bestimme Koeff. vor p_(k-1)
  cc(k-1,1) = h(k-1)/h(k-2);      % bestimme Koeff. vor p_(k-2)
  p{k} = [0;p{k-1}] - aa(k-1)*[p{k-1};0] - cc(k-1) * [p{k-2};0;0];
end
if length(aa)==1                  % sym. Matrix A aufstellen
  A = aa;
else
  A = full(spdiags([[sqrt(cc(2:end));0],aa,sqrt(cc)],...
      [-1 0 1],n-1,n-1));
end
[V,D] = eig(A);                   % Eigenwerte bestimmen
nodes = diag(D);                  % Quad.-stellen sind Eigenw. von A
weights = zeros(n-1,1);           % Quadraturgewichte sind skalierte
for k = 1:n-1                     % Skalarprodukte der Eigenvektoren
  weights(k) = momente(1) * V(1,k)ˆ2 / (V(:,k)' * V(:,k));
end
kn = integ(mult(p{n},p{n}),momente); % Koeffizient im Fehlerterm
% einige Unterfunktionen zur Behandlung der auftretenden Polynome
% Polynome miteinander multiplizieren
  function p = mult(s,t)
    p = zeros(length(s)+length(t)-1,1);
    for j = 1:length(s);
      p(j:length(t)+j-1) = p(j:length(t)+j-1) + s(j) * t(:);
    end
  end
% Polynome addieren
  function r = add(p,q)
    if length(q) >= length(p)
      r = q(:);
      r(1:size(p,1)) = r(1:size(p,1)) + p(:);
    else
      r = p(:);
      r(1:size(q,1)) = r(1:size(q,1)) + q(:);
    end
  end
% Integral auswerten
  function r = integ(p,momente)
    r = (p(:))'*momente(1:length(p))';
  end
end
```

MATLAB-Funktion: GaussQuadratur.m

```
function [weights,nodes,kn] = QaussQuadratur(anz_Gewichte,typ)
  N = 2 * anz_Gewichte+1;
  switch typ
    case 'legendre'
      momente = 2./(1:N); momente(2:2:end) = 0;
    case 'tschebyscheff'
      momente = [pi,0,pi/2];
      for j= 2:anz_Gewichte
        momente = [momente,0,momente(end)*(2*j-1)/(2*j)];
      end
    case 'laguerre'
      momente = factorial(0:N-1);
    case 'hermite'
      momente = gamma((1:N)./2); momente(2:2:end) = 0;
    case 'log'
      momente = (1./(1:N)).^2;
    otherwise
      disp('Nur "legendre", "tschebyscheff", ', ...
           '"laguerre", "hermite", "log" möglich!')
      return
  end
  [weights,nodes,kn] = GaussWeightsNodes(momente);
end
```

Man kann z.B. die Quadraturstellen und Gewichte zur Gewichtsfunktion

$$\omega(x) = -\log(x)$$

und $n = 2$ berechnen lassen, sie erfüllen dann

$$-\int_0^1 \log(x) f(x)\,dx = \sum_{j=1}^{n} \lambda_j f(x_j) + \frac{f^{(2n)}(\xi)}{(2n)!} K_n\,,$$

mit $K_n = (p_n, p_n)_\lambda / k_n^2$.

MATLAB-Beispiel:

```
>> [x,w,Kn] = GaussQuadratur(2,'log')
x =
    0.7185
    0.2815
w =
    0.1120
    0.6023
Kn =
  0.0029
```

Bemerkung 3.4.36 *Umfangreiche Tabellen von Stützstellen und Gewichten zu verschiedenen Gewichtsfunktionen findet man z.B. in [AS] und [St].*

Tab. 3.1: Momente für einige Standardfälle von Orthogonalpolynomen

Name	a	b	$\lambda(x)$	$m_k := \int_a^b \lambda(x)x^k\,dx$
Legendre	-1	1	1	$m_k = \begin{cases} 2/(k+1) & \text{falls } k \text{ gerade} \\ 0 & \text{falls } k \text{ ungerade} \end{cases}$
Tscheby.	-1	1	$\frac{1}{\sqrt{1-x^2}}$	$m_k = \begin{cases} \pi \cdot \frac{1\cdot 3\cdot 5\cdot \ldots \cdot k-1}{2\cdot 4\cdot 6\cdot \ldots \cdot k} & \text{falls } k \text{ gerade} \\ 0 & \text{falls } k \text{ ungerade} \end{cases}$
Laguerre	0	∞	e^{-x}	$m_k = k!$
Hermite	$-\infty$	∞	e^{-x^2}	$m_k = \begin{cases} \Gamma(\frac{k+1}{2}) & \text{falls } k \text{ gerade} \\ 0 & \text{falls } k \text{ ungerade} \end{cases}$
Jacobi $\alpha = \beta > -1$	-1	1	$(1-x)^\alpha(1+x)^\beta$	$m_k = \begin{cases} \sqrt{\pi}\frac{1\cdot 3\cdot 5\cdot \ldots \cdot k-1}{2^{k/2}} \frac{\Gamma(\alpha+1)}{\Gamma(\alpha+(k+3)/2)} & \text{falls } k \text{ gerade} \\ 0 & \text{falls } k \text{ ungerade} \end{cases}$
„log“	0	1	$-\log(x)$	$m_k = 1/(k+1)^2$

Bemerkung 3.4.37 (Gauß-Qadratur in höheren Dimensionen) *In höheren Dimensionen hat man eine solche Theorie der Orthogonalpolynome nicht, aus der man Quadraturformeln gewinnen könnte. Hier bleibt einem häufig nichts anderes übrig, als diese näherungsweise als Lösung nichtlinearer Gleichungen zu bestimmen.*

3.5 Schwierigkeiten bei der Quadratur

In diesem Abschnitt diskutieren wir einige Schwierigkeiten bzw. den möglichen Umgang mit schwierige Integranden.

3.5.1 Unstetige Integranden

Häufig ist der Integrand nur durch Punktauswertungen bekannt, und als Komposition mehrerer Teilfunktionen definiert, die an den Nullstellen unstetig sind. Zum Beispiel ist eine Funktion auf unterschiedlichen Teilintervallen, durch unterschiedliche Approximationen definiert, wie z.B. die *Besselfunktion*[8]. Sind diese Nahtstellen bekannt, so sollte man das Integrationsgebiet hier unterteilen, da kommerzielle Programme als auch die später noch diskutierte adaptive Quadratur hier versagen oder ineffizient sind.

3.5.2 Singuläre Integrale

Definition 3.5.1 (Singuläres Integral) *Unter einer singulären Funktion wollen wir eine Funktion verstehen, die mit Ausnahme endlich vieler Stellen auf einem Intervall $[a, b]$ definiert und stetig ist, sodass f in jeder Umgebung einer Unstetigkeitsstelle unbeschränkt ist.*

Beispiel 3.5.2 Zum Beispiel ist die Funktion $1/\sqrt{x}$ auf $[0, 1]$ unbeschränkt und trotzdem besitzt

[8] Bessel, Friedrich Wilhelm (1784-1846)

sie endliches Integral, nämlich

$$I(f) = \int_0^1 \frac{1}{\sqrt{x}}\, dx = 2$$

Betrachten wir nun mehrere Methoden, wie sich solche Integrale doch noch in „angenehmere“ Integrale umschreiben lassen:

a) **Substitution**
Für ein $x_0 \in [a,b]$ mögen die einseitigen Grenzwerte der Ableitungen nicht existieren, z.B. $f(x) = \sqrt{x}\sin(x)$ auf $[0,1]$ und $x_0 = 0$. Hier läßt sich schon die zweite Ableitung $f''(0)$ nicht definieren. Die Substitution $t := \sqrt{x}$ führt hier zum Ziel

$$\int_0^1 \sqrt{x}\sin(x)\, dx = \int_0^1 2t^2 \sin(t^2)\, dt.$$

Der Integrand ist nun sogar beliebig oft differenzierbar.

b) **Regularisierung**
Eine Idee, die unter dem Stichwort *Abziehen der Singularität* oder *Regularisierung* bekannt ist, besteht aus der Anwendung der Formel

$$I(f) = I(f-s) + I(s).$$

Im obigen Beispiel leistet das die Funktion $s = \sqrt{x}\cdot x$:

$$\begin{aligned}\int_0^1 \sqrt{x}\sin(x)\, dx &= \int_0^1 \sqrt{x}\sin(x) - \sqrt{x}\cdot x\, dx + \int_0^1 \sqrt{x}\cdot x\, dx \\ &= \int_0^1 \sqrt{x}\big(\sin(x) - x\big)\, dx + \frac{2}{5}\end{aligned}$$

Der neue Integrand ist jetzt dreimal stetig differenzierbar. Man berücksichtige jedoch, dass nun Nahe der Stelle $x_0 = 0$ für kleine x Auslöschungen auftreten können.

c) **Aufspaltung des Integrals**
Eine weitere Möglichkeit besteht manchmal in der Aufspaltung des Integrals, im Fall $a)$ wäre dies

$$\int_0^1 \sqrt{x}\sin(x)\, dx = \int_0^\varepsilon \sqrt{x}\sin(x)\, dx + \int_\varepsilon^1 \sqrt{x}\sin(x)\, dx \quad 0 < \varepsilon < 1$$

Der zweite Integrand ist nun glatt, wobei die Konstante der n-ten Ableitung von ε abhängt. Für das erste Integral erhält man, wenn man $\sin(x)$ in eine Reihe entwickelt

$$\begin{aligned}\int_0^\varepsilon \sqrt{x}\sin(x)\, dx &= \int_0^\varepsilon \sqrt{x}\sum_{k=0}^{\infty} \frac{x^{2k+1}}{(2k+1)!}(-1)^k \\ &= \sum_{k=0}^{\infty}(-1)^k \frac{\varepsilon^{2k+5/2}}{(2k+1)!(2k+5/2)}.\end{aligned}$$

Dieses Vorgehen ist gerechtfertigt, da man wegen der gleichmäßigen Konvergenz der Reihe diese gliedweise ausrechnen darf, bzw. Summation und Integration vertauschen darf. Für ein kleines ε kann man diese Entwicklung nach wenigen Schritten abbrechen. Das Problem hierbei ist die geeignete Wahl des ε.

d) **Anwendung der *Gauß-Quadratur***
Ein weiterer Kunstgriff besteht darin, dass man das Integral in die Form

$$I(f) = \int_a^b \omega(x) f(x)\, dx$$

bringt, wobei $\omega(x)$ die Singularität enthält. Zur Integration zieht man dann eine *Gauß*-Formel zum Gewicht $\omega(x)$ heran. Erneut berechnen wir das oben verwendete Integral

$$\int_0^1 \sqrt{x}\sin(x)\, dx$$

mit Hilfe einer *Gauß*-Formel zum Gewicht $\sqrt{x}$ für $x \in (0,1]$. Die *Jacobi*-Polynome $P_n^{0,1/2}$ sind gerade die bezüglich $\omega(x) = (1+x)^{1/2}$ auf $(-1,1)$ orthogonalen Polynome. Mit den Überlegungen zu den *Gauß*-Gewichten lassen sich

$$m_k = \int_0^1 \sqrt{x} x^k\, dx = \frac{2}{2k+3}$$

Quadraturpunkte und Gewichte bestimmen.

3.6 Numerische Quadratur von stark oszillierenden Integranden

Motivation Die Integration von stark oszillierenden Integranden ist ein numerisches Problem von weitreichender Bedeutung mit einer Vielzahl von Anwendungen, z.B. in der Quantenchemie, Bildanalyse, Elektrodynamik, Computertomographie und Strömungsmechanik. Allgemein wird dieses Problem als „schwierig“ eingestuft, bei dem man am besten die Oszillationen eliminiert, z.B. durch die Wahl überaus vieler kleiner Teilintervalle. Da das folgende Kapitel nur einen Einblick in diese Problematik und Techniken liefern soll, beschränken wir uns auf ein Integral der Form

$$I(f) = \int_0^1 f(x) e^{i\omega g(x)} dx \tag{3.57}$$

Bemerkung 3.6.1 *Man beachte, dass sich ein beliebiges Integral $\int_a^b \widetilde{f}(y) e^{i\omega\widetilde{g}(y)} dy$ mit der Transformation $y = a + (b-a)x$ auf die Form von* (3.57) *überführen läßt, dabei ist dann $f(x) = \widetilde{f}(a+(b-a)y)/(b-a)$ und $g(x) = \widetilde{g}(a+(b-a)y)$.*

Bemerkung 3.6.2 *Das übliche numerische Verfahren ein Integral der Form* (3.57) *zu bestimmen, wobei f und g als glatt vorausgesetzt werden, wäre die* Gauß*-Quadratur: Wir interpolieren den Integranden an paarweise verschiedenen Knoten $0 \le x_0 < \ldots < x_n \le 1$ durch ein Polynom $P \in \mathbb{P}_n$ und approximieren*

$$I(f) \approx \int_0^1 P(x)\, dx\,.$$

Unglücklicherweise liefert diese Vorgehensweise für $w \gg 1$ völlig sinnlose Ergebnisse.

Ein alternativer Ansatz stammt in seiner ersten Idee von *Louis N. G. Filon* (1928). Anstatt den ganzen Interpolanden $f(x)e^{i\omega g(x)}$ zu interpolieren, interpolieren wir an den Stellen $x_0, \ldots, x_n$ die Funktion $f(x)$ durch ein Polynom $P(f|x_0,\ldots,x_n)(x)$ und setzen

$$\hat{I}^{Filon}(f) = \int_0^1 P(f|x_0,\ldots,x_n)(x) e^{i\omega g(x)} dx = \sum_{k=0}^n \lambda_k(\omega) f(x_k),$$

wobei $\lambda_k(\omega) := \int_0^1 L_k^{(n)}(x)e^{i\omega g(x)}dx$ und $L_k^{(n)}(x)$ sei das *Lagrange*-Polynom zu den Knoten $x_0, \ldots, x_n$. Man beachte dabei, dass die Konstruktion von $\hat{I}^{Filon}$ voraussetzt, dass sich die ersten Momente $\int_0^1 x^m e^{i\omega g(x)}dx$ berechnen lassen. Dies setzen wir stillschweigend für den Rest dieses Abschnitts voraus. Falls $g' \neq 0$ in $[0,1]$, läßt sich zeigen (vgl. *Iserles*, 2003a), dass eine größer werdende Frequenz ω zu kleineren Fehlern führt, genauer ausgedrückt

$$\hat{I}^{Filon}(f) - I(f) \sim \begin{cases} \mathcal{O}(\omega^{-1}), & , \quad x_0 > 0 \text{ oder } x_n < 1 \\ \mathcal{O}(\omega^{-2}), & , \quad x_0 = 0 \text{ und } x_n = 1 \end{cases} \quad \omega \to \infty$$

Diese Methode läßt sich noch dadurch verbessern, dass man das Interpolationspolynom durch ein allgemeines *Hermite*-Polynom ersetzt, also ein Polynom wählt, welches an den Stellen $x_0, \ldots, x_n$ nicht nur die Funktionswerte $f(x_0), \ldots, f(x_n)$, sondern auch die Ableitungen $f^{(m)}(x)$ an den Stützstellen exakt repräsentiert. Durch diese Verallgemeinerung läßt sich auch die Einschränkung $g' \neq 0$ in $[0,1]$ umgehen, falls g' an endlich vielen Stellen ξ_k mit $g'(\xi_k) = \ldots = g^{(r_k)}(\xi_k)$ und $g^{(r_k+1)}(\xi_k) \neq 0$ gilt.

Beispiel 3.6.3 Wie oben sei $f(x) = \cos(x), g(x) = x$ und $\omega > 0$. Das Polynom

$$P(x) = \sum_{k=0}^{n} f(x_k) \cdot \Big(1 - \frac{\omega''(x_n)}{\omega'(x_k)}(x - x_k)\Big) L_k^2(x) = \sum_{k=0}^{n} f'(x_k) \cdot (x - x_k)(L_k^{(n)}(x))^2,$$

wobei $L_k^{(n)}$ die *Lagrange*-Polynome sind und $\omega(x) = (x - x_0) \cdot \ldots \cdot (x - x_n)$ ist, erfüllt das *Hermite'sche* Interpolationsproblem

$$P(x_k) = f(x_k), \quad P'(x_k) = f'(x_k), \quad k = 0, 1, 2, \ldots, n;$$

für $n = 1$ lautet

$$\begin{aligned} P(x) &= \frac{(3x_0 - x_1 - 2x)(x - x_1)^2}{(x_0 - x_1)^2} f(x_0) + \frac{(3x_1 - x_0 - 2x)(x - x_0)^2}{(x_1 - x_0)^2} f(x_1) \\ &\quad + \frac{(x - x_0)(x - x_1)^2}{(x_0 - x_1)^2} f'(x_0) + \frac{(x - x_1)^2(x - x_2)}{(x_2 - x_1)^2} f'(x_2). \end{aligned}$$

Setzen wir $x_0 = 0$ und $x_1 = 1$, so erhalten wir

$$P(x) = (1 + 2x)(x - 1)^2 f(0) + (3 - 2x)x^2 f(1) + x(x - 1)^2 f'(0) + x^2(x - 1)f'(1).$$

Mit Maple lassen sich leicht die Integrale

$$\int_0^1 (1 + 2x)(x - 1)^2 e^{i\omega x}dx, \ldots, \int_0^1 x^2(x - 1)e^{i\omega x}dx$$

berechnen, wobei die entsprechenden Zeilen in Maple lauten

```
> collect(int((1+2*x)*(x-1)^2*exp(I*omega*x)  ,x=0..1), omega);
  collect(int((3-2*x)*x^2      *exp(I*omega*x)  ,x=0..1), omega);
  collect(int(x*(x-1)^2        *exp(I*omega*x)  ,x=0..1), omega);
  collect(int(x^2*(x-1)        *exp(I*omega*x)  ,x=0..1), omega);
```

Die entsprechende Quadraturformel lautet dann

$$\begin{aligned} \hat{I}^{Filon}(f) &= \Big(\frac{i}{\omega} + \frac{6i(1 + e^{i\omega})}{\omega^3} + \frac{12(1 - e^{i\omega})}{\omega^4}\Big) f(0) \\ &+ \Big(\frac{-ie^{i\omega}}{\omega} - \frac{6(1 + e^{i\omega})}{\omega^3} - \frac{12(1 - e^{i\omega})}{\omega^4}\Big) f(1) \\ &+ \Big(-\frac{1}{\omega^2} + \frac{2i(2 + e^{i\omega})}{\omega^3} + \frac{6(1 - e^{i\omega})}{\omega^4}\Big) f'(0) \\ &+ \Big(\frac{e^{i\omega}}{\omega^2} + \frac{2i(1 + 2e^{i\omega})}{\omega^3} + \frac{6(1 - e^{i\omega})}{\omega^4}\Big) f'(1) \end{aligned}$$

Für $f(x) = \cos(x), g(x) = x$ und $10 < \omega < 100$ ist der Fehler für Q^{Filon} skaliert mit ω^3 in der nachfolgenden Abbildung dargestellt.

Betrachten wir nun die weitere Möglichkeit das Integral (3.57) mittels einer asymptotischen Entwicklung zu berechnen. Aus

$$\frac{d}{dx}e^{i\omega g(x)} = i\omega g'(x)e^{i\omega g(x)}$$

erhalten wir nun unter der Voraussetzung $g' \neq 0$ in $[0,1]$, $\omega > 0$ für eine beliebige Funktion h mittels partieller Integration

$$\begin{aligned}\int_0^1 h(x)e^{i\omega g(x)}dx &= \int_0^1 \frac{h(x)}{i\omega g'(x)} i\omega g'(x)e^{i\omega g(x)}dx \\ &= \frac{h(x)}{i\omega g'(x)} \cdot e^{i\omega g(x)}\Big|_{x=0}^1 - \int_0^1 \frac{1}{i\omega}\frac{d}{dx}\Big[\frac{h(x)}{g'(x)}\Big]e^{i\omega g(x)}dx \qquad (3.58)\end{aligned}$$

Mit der Notation

$$\sigma_0[f](x) = f(x), \quad \sigma_{k+1}[f](x) = \frac{d}{dx}\frac{\sigma_k[f](x)}{g'(x)}, \quad k = 0, 1, 2, \ldots$$

und (3.58) erhalten wir somit für $h(x) = \sigma_k[f](x)$:

$$\begin{aligned}\int_0^1 \sigma_k[f](x)e^{i\omega g(x)} &= \frac{1}{i\omega}\Big(\frac{\sigma_k[f](1)}{g'(1)}e^{i\omega g(1)} - \frac{\sigma_k[f](0)}{g''(0)}e^{i\omega g(0)}\Big) \\ &- \frac{1}{i\omega}\int_0^1 \sigma_{k+1}[f](x)e^{i\omega g(x)}dx.\end{aligned}$$

Per Induktion folgt nun für $n \geq 0$

$$\begin{aligned}\int_0^1 f(x)e^{i\omega g(x)}dx &= -\sum_{l=1}^n \Big(\frac{i}{\omega}\Big)^l \Big(\frac{e^{i\omega g(1)}}{g'(1)} \cdot \sigma_{l-1}[f](1) - \frac{e^{i\omega g(0)}}{g'(0)} \cdot \sigma_{l-1}[f](0)\Big) \\ &+ \Big(\frac{i}{\omega}\Big)^n \int_0^1 \sigma_n[f](x)e^{i\omega g(x)}dx\end{aligned}$$

Die Idee der asymptotischen Reihenentwicklung ist nun, das Integral $(\frac{i}{\omega})^n \int_0^1 \sigma_n[f](x)e^{i\omega g(x)}dx$ wegzulassen, da $\frac{1}{\omega^n}$ für $\omega \gg$ „hoffentlich schneller “gegen Null strebt, als das entsprechende Integral. Somit definieren wir die Quadratur

$$\hat{I}^{asymp.}(f) = -\sum_{l=1}^n \Big(\frac{i}{\omega}\Big)^l \Big(\frac{e^{i\omega g(1)}}{g'(1)} \cdot \sigma_{l-1}[f](1) - \frac{e^{i\omega g(0)}}{g'(0)} \cdot \sigma_{l-1}[f](0)\Big)$$

Für streng monotone g erhalten wir durch direktes Ausrechnen

$$\begin{aligned}\sigma_0[f] &= f \\ \sigma_1[f] &= -\frac{g''}{g'^2}f + \frac{1}{g'}f' \\ \sigma_2[f] &= \frac{3g''^3 - gg'''}{g'^4}f - \frac{3g''}{g'^3}f' + \frac{1}{g'^2}f'' \\ \sigma_3[f] &= \frac{-15g''^3 + 10g'g''g''' - g^2g^{(4)}}{g'^6}f + \frac{15g''^2 - 4gg'''}{g'^5}f' - \frac{6g''}{g'^4}f'' + \frac{1}{g'^3}f'''\end{aligned}$$

und man sieht sofort, dass für jedes $\ell \geq 0$ und $j = 0, 1, \ldots, \ell$ Funktionen $\sigma_{\ell,j}$ existieren, welche von g und entsprechenden Ableitungen abhängen, sodass

$$\sigma_\ell[f](x) = \sum_{j=0}^{\ell} \sigma_{\ell,j} f^{(j)}(x) \quad \text{und } \sigma_{\ell,\ell} = \frac{1}{(g'(x))^\ell} \neq 0$$

Für unser Beispiel lautet die Quadraturformel $\hat{I}^{app}$ mit 2 Entwicklungstermen

$$\hat{I}^{asymp.}(f) = \frac{e^{i\omega} f(1) - f(0)}{i\omega} + \frac{e^{i\omega} f'(1) - f'(0)}{\omega^2}.$$

Für $f(x) = \cos(x), g(x) = x$ und $10 < \omega < 100$ ist der Fehler für $\hat{I}^{asymp.}$ skaliert mit ω^3 in nachfolgender Grafik dargestellt (oberer Graph)

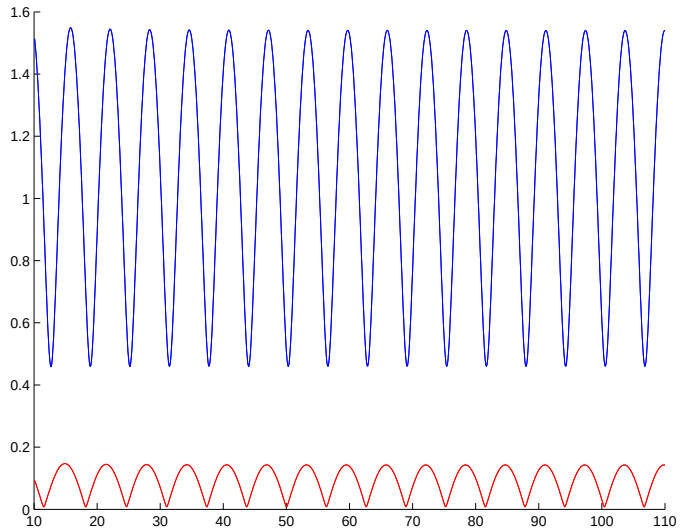


Abb. 3.5: Der Fehler von $\hat{I}^{Filon}$ (oben) und $\hat{I}^{asymp.}$ (unten), skaliert mit ω^3 für $f(x) = \cos(x), g(x) = x$ und $10 < \omega < 100$.

Widmen wir uns noch der Einschränkung, dass $g' \neq 0$ in $[0, 1]$ gelten soll, was wir in unseren Betrachtungen immer vorausgesetzt haben. Man beachte nun, dass für jedes $\xi \in [0, 1]$ und glatte Funktionen f und g mit $g'(\xi) = 0, g''(\xi) \neq 0$ und $g'(x) \neq 0$ für $x \in [0, 1] \backslash \{\xi\}$ die folgende Gleichheit gilt

$$\begin{aligned} \int_0^1 f(x) e^{i\omega g(x)} dx &= f(\xi) \int_0^1 e^{i\omega g(x)} dx + \int_0^1 \big(f(x) - f(\xi)\big) e^{i\omega g(x)} dx \\ &= f(\xi) \int_0^1 e^{i\omega g(x)} dx + \frac{1}{i\omega} \int_0^1 \frac{f(x) - f(\xi)}{g'(x)} \cdot \frac{d}{dx} e^{i\omega g(x)} dx \end{aligned}$$

Mit dem Satz von *de l'Hospital*[9] folgt für genügend glatte f und g nun, da $g'' \neq 0$ in $[0, 1]$, dass der Grenzwert

$$\lim_{x \to \xi} \frac{f(x) - f(\xi)}{g'(x)} = \lim_{x \to \xi} \frac{f'(x)}{g''(x)}$$

existiert. Mit der Notation

$$\begin{aligned} \rho_o[f](x) &= f(x) \\ \rho_{k+1}[f](x) &= \frac{d}{dx} \frac{\rho_k[f](x) - \rho_k[f](\xi)}{g'(x)}, \quad k \geq 0 \end{aligned}$$

[9] de l'Hospital (1661-1704)

folgt dann wieder mittels partieller Integration und Induktion

$$\int_0^1 f(x)e^{i\omega g(x)}dx = \int_0^1 e^{i\omega g(x)}dx \sum_{\ell=0}^{n-1}\left(\frac{i}{\omega}\right)^\ell \rho_\ell[f](\xi)$$
$$-\sum_{\ell=1}^{n}\left(\frac{i}{\omega}\right)^\ell\left(\frac{e^{i\omega g(1)}}{g'(1)}\Big\{\rho_{\ell-1}[f](1)-\rho_{\ell-1}[f](\xi)\Big\}-\frac{e^{i\omega g(0)}}{g'(0)}\Big\{\rho_{\ell-1}[f](0)-\rho_{\ell-1}[f](\xi)\Big\}\right)$$
$$+\left(\frac{i}{\omega}\right)^n\int_0^1 \rho_n[f](x)e^{i\omega g(x)}dx$$

Beispiel 3.6.4

$$\int_{-1}^{1}\cos(x*\exp(4*x^2))\,dx = 2\Re\left(\int_0^1 \exp((2x-1)*\exp(4*(2x-1)^2))\,dx\right)$$

MATLAB-Funktion: asymp_bsp.txt

```
> restart:
> f := x -> 1:
> g := x -> (2*x-1) * exp(4*(2*x-1)^2):
> rho := proc(k)
>   local i,tmp;
>   tmp := f(x);
>   for i from 1 to k do
>     tmp:=diff(tmp/diff(g(x),x),x);
>   end do;
>   simplify(tmp)
> end proc:
>
> simplify(subs(x=(y+1)/2,rho(4)))
```

$$\begin{aligned}
\rho_0[f](x) &= 1\\
\rho_1[f](x) &= \left.\frac{-8\exp(-4y^2)y(8y^2+3)}{(8y^2+1)^2}\right|_{y=2x-1}\\
\rho_2[f](x) &= \left.\frac{8\exp(-8y^2)(1024y^6+704y^4+144y^2-3)}{(8y^2+1)^4}\right|_{y=2x-1}\\
\rho_3[f](x) &= \left.\frac{-8\exp(-12y^2)y(24576y^8+24064y^6+9024y^4+1160y^2-75)}{(8y^2+1)^6}\right|_{y=2x-1}
\end{aligned}$$

4 SPLINES

In diesem Kapitel wird die Theorie polynomialer und rationaler Splinefunktionen dargestellt. Zuerst stellt sich jedoch die Frage, warum überhaupt die Spline-Approximation eingeführt wurde. Ein kleiner Ausflug in die Geschichte fördert Interessantes zu Tage.

Bemerkung 4.0.1 (Historischer Hintergrund der Spline-Interpolation) *Das Wort „Spline“ bedeutet etwa „dünne Holzlatte“. Im Schiffsbau etwa seit dem 18. Jahrhundert wurden solche Holzlatten um die Spanten gelegt, um so einen Schiffsrumpf aus Holz zu formen (vgl. Abbildung 4.1).*

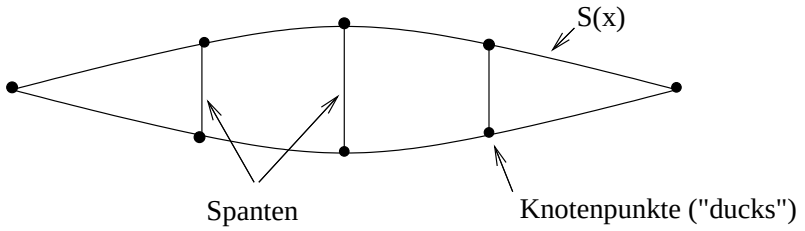


Abb. 4.1: Schematische Darstellung eines Schiffsrumpf

Die Latte biegt sich so, dass die Biegeenergie minimal wird. Die „Biegeenergie“E eines elastischen Körpers ist gegeben durch

$$E = \int_a^b \left(\frac{s''(x)}{(1+s'(x)^2)^{3/2}} \right)^2 dx,$$

also dem Integral über das Quadrat der Krümmung

$$\kappa(x) = \frac{s''(x)}{(1+s'(x)^2)^{3/2}}.$$

Bei schlanken Booten ist die Biegung s' beschränkt, d.h. wir haben kleine Auslenkungen. Für kleine $s'(x)$ liefert $s''(x)$ eine vernünftige Näherung an $\kappa(x)$

$$E \approx \int_a^b s''(x)^2\, dx = \|s''(x)\|_2^2.$$

also

$$\min E \quad u.d.N. \quad s(x_i) = f_i, \quad i = 1, \ldots, n$$

wobei (x_i, f_i) die gegebenen Daten sind. Wir werden sehen, dass unter allen Interpolierenden ein Spline das Integral über das Quadrat der Krümmung „minimiert“, ein Spline also die „glatteste“ Interpolationsfunktion.

Im Gegensatz zur Polynominterpolation verhindert das „Glattheitsmaß“ das Oszillieren. *Euler* und *Bernoulli* haben schon erkannt, dass die Lösung s des Minimierungsproblems folgende Eigenschaften besitzt

1. $s|_{[x_i, x_{i+1})} \in \mathbb{P}_3$ (lokal polynomial),
2. $s \in C^2[a, b]$ (global glatt).

Dies wird uns zur Definition von Splines führen. Präzise formulieren kann man die Approximationseigenschaften von Spline wie folgt:

Satz 4.0.2 (von Jackson, vgl. [Rivlin], Seite 23, in anderer Form [Schönhage], Seite 182) *Seien $-\infty < a < b < \infty, k \in \mathbb{N}$ und $f \in C^k([a,b])$ sowie $n < k$. Dann existieren $c_k \in \mathbb{R}^+$ mit*

$$E_n(f) \leq c_k \left(\frac{b-a}{n}\right)^k \omega\left(f^{(k)}, \frac{b-a}{2(n-k)}\right),$$

wobei

$$E_n(f) := dist(f, \mathbb{P}_n) = \inf_{g \in \mathbb{P}_n} \|f - g\|_{\infty,[a,b]} \quad \textit{und}$$

der Fehler der Bestapproximation und

$$\omega(g,h) := \sup_{|\delta|<h} \|g(\cdot) - g(\cdot + \delta)\|_{\infty,[a,b]}$$

der Stetigkeitsmodul von g ist, ein Maß für die Glattheit von g.

Der Satz von Jackson besagt folgendes: Ist f „glatt“ in dem Sinne, dass das Stetigkeitsmodul ω „klein“ ist, dann ist auch die beste Approximation durch Polynome „gut“. Ist f hingegen nicht „glatt“ , kann man nicht unbedingt hoffen, f „gut“ durch Polynome approximieren zu können. Ferner treten folgende Polynome auf,

- die Interpolationspolynome sind abhängig von der Wahl der Knoten, wie schon das Beispiel von Runge (siehe Seite 58) zeigt,
- bei Änderung eines Koeffizienten eines Polynoms tritt eine globale Änderung ein, und
- viele Basen sind schlecht konditioniert.

Die einfachste Form einer stetigen Funktion f, die die Bedingung $f(x_i) = y_i$ für gegebene geordnete Paare (x_i, y_i) $(i = 0, \ldots, n)$ erfüllt, ist sicherlich der Streckenzug, d.h.

$$f|_{(x_{i-1},x_i)} = \frac{y_i - y_{i-1}}{x_i - x_{i-1}}(\cdot - x_{i-1}) + y_{i-1}, \quad i = 1, \ldots, n\,.$$

Auf jedem Intervall ist f also ein Polynom höchstens ersten Grades und insgesamt eine stetige Funktion. Dies wollen wir nun verallgemeinern.

Definition 4.0.3 (Splineraum $\mathcal{S}^k(\mathcal{T})$) *Es sei $\mathcal{T} = \{x_0, \ldots, x_n\}$ eine Knotenfolge von $n+1$ paarweise verschiedenen Knoten*

$$a = x_0 < \cdots < x_n = b\,.$$

*Ein **Spline** vom Grad k, $k \geq 0$, bezüglich $\mathcal{T}$ ist eine Funktion $s \in C^{k-1}[a,b]$, für die auf jedem Intervall $[x_i, x_{i+1}]$, $i = 0, \ldots, n-1$*

$$s|_{[x_i,x_{i+1}]} \in \mathbb{P}_k$$

gilt. Den **Raum aller Splines vom Grad k zur Knotenfolge** *$\mathcal{T}$ bezeichnen wir mit $\mathcal{S}^k(\mathcal{T})$. Unter $C^{-1}[a,b]$ ist hierbei der Raum der stückweise stetigen Funktionen zu verstehen, d.h. unstetig nur an den Knoten x_i, $i = 0, \ldots, n$.*

Bemerkung 4.0.4 *Die vorstehende Definition der Splineräume entspricht der in [Quateroni] S.23. Man beachte allerdings, dass in anderen Lehrbüchern wie z.B. [Deufelhard] eine andere Definition zugrunde liegt.*

4.1 KUBISCHE SPLINE-INTERPOLATION

Im Folgenden betrachten wir die Interpolation mit **kubischen Splines**.

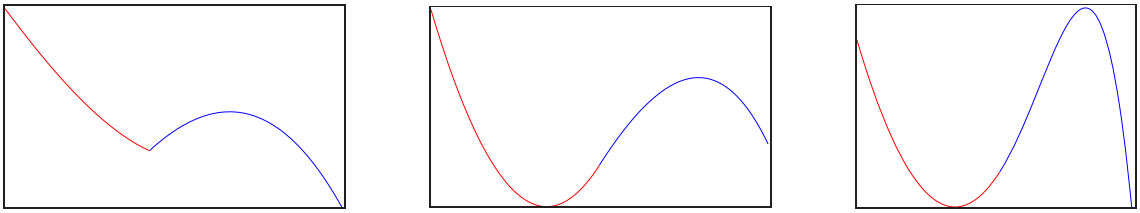


Abb. 4.2: Verschiedene Funktionen aus C^0, C^1 und C^2

Welche dieser Funktionen in der Sequenz 4.2 von Grafiken empfinden Sie als glatt?

Der Knick im linken Graphen ist offensichtlich, auch im mittleren Graphen erkennt man mit etwas Geduld und Erfahrung eine Unstetigkeit in der Krümmung der Funktion, welche jedoch in der rechten Grafik nicht mehr auszumachen ist.

In vielen grafischen Anwendungen genügen diese Glattheitsanforderungen an die Interpolationsfunktion, nämlich sie vom Auge als „glatt“ zu empfinden. Dies ist einer der Hauptgründe, weswegen wir uns zuerst auf Funktionen aus $\mathcal{S}^3(\mathcal{T}) \subset C^2$ beschränken. Der Vollständigkeit halber definieren wir:

Definition 4.1.1 (Kubischer Spline) *Eine Funktion $s : [a,b] \to \mathbb{R}$ heißt* **kubischer Spline** *bezüglich einer Zerlegung $\mathcal{T}$, falls $s \in S^3(\mathcal{T})$, d.h. wenn gilt*

1. $s|_{[x_i,x_{i+1})} \in \mathbb{P}_3, \quad i = 0,\dots,n-1$ *(lokal polynomial),*
2. $s \in C^2[a,b]$ *(global glatt).*

Konstruktion kubischer Splines: Geben seien geordnete Datenpaare $(x_0,y_0),\dots,(x_n,y_n)$ mit $y_i := f(x_i)$ und eine durch diese induzierte Knotenfolge $\mathcal{T} = \{x_0,\dots,x_n\}$, die das Intervall $[a,b]$ in n Teilintervalle zerlegt.

Für das i-te Teilintervall $I_i := [x_i, x_{i+1}]$ der Länge $h_i := x_{i+1} - x_i$ wählen wir folgenden **Ansatz**:

$$s_i(x) := s(x)\big|_{[x_i,x_{i+1}]} = a_i(x-x_i)^3 + b_i(x-x_i)^2 + c_i(x-x_i) + d_i. \tag{4.1}$$

Bei n Teilintervallen haben wir also $4n$ unbekannte Koeffizienten a_i, b_i, c_i, d_i zu bestimmen. Damit die aus den lokalen Polynomen s_i vom Grad kleiner gleich 3 zusammengesetzte Funktion s global zweimal stetig differenzierbar ist (vgl. Definition 4.1.1), müssen wir zusätzlich zu den Interpolationsbedingungen Stetigkeits und-Differenzierbarkeitsbedingungen fordern.

Die **Interpolationsbedingungen**

$$s_i(x_i) = f(i), \quad s_i(x_{i+1}) = f_{i+1}, \quad i = 0,1,\dots,n-1,$$

ergeben $2n$ Gleichungen, welche die Stetigkeit von s sicher stellen.

Die **Stetigkeit der ersten Ableitung an den inneren Knoten**, d.h. $s \in C^1[a,b]$,

$$s'_{i-1}(x_i) = s'_i(x_i), \quad i = 1,\dots,n-1,$$

liefert $n-1$ zusätzliche Gleichungen. Und die **Stetigkeit der zweiten Ableitung an den inneren Knoten**, d.h. $s \in C^2[a,b]$ ergibt noch mal $n-1$ Gleichungen

$$s''_{i-1}(x_i) = s''_i(x_i), \quad i = 1, \ldots, n-1.$$

Zusammen ergibt dies $(2n+2(n-1))$ Bedingungen. Die $4n-(2n+2(n-1)) = 2$ verbleibenden Freiheitsgrade werden durch **Randbedingungen** festgelegt, d.h. üblicher Weise durch zusätzliche Bedingungen an die ersten oder zweiten Ableitungen von s an den Intervallgrenzen a und b.

Bemerkung 4.1.2 (Dimension von $\mathcal{S}^3(\mathcal{T})$) *Bevor wir auf die Eigenschaften der kubischen Splines eingehen, noch eine Anmerkung zur **Dimension von** $\mathcal{S}^3(\mathcal{T})$ mit $\mathcal{T} = \{x_0, \ldots, x_n\}$. Für das i-te Teilintervall $I_i := [x_i, x_{i+1}]$ der Länge $h_i := x_{i+1} - x_i$ wählen wir den Ansatz (4.1) für s_i. Für seinen Wert und die Ableitungen s', s'' an den Endpunkten erhalten wir*

$$\begin{aligned}
s_i(x_i) &= d_i = f(x_i), && (4.2)\\
s_i(x_{i+1}) &= a_i h_i^3 + b_i h_i^2 + c_i h_i + d_i = f(x_{i+1}), && (4.3)\\
s_i'(x_i) &= c_i, && (4.4)\\
s_i'(x_{i+1}) &= 3a_i h_i^2 + 2b_i h_i + c_i, && (4.5)\\
s_i''(x_i) &= 2b_i, && (4.6)\\
s_i''(x_{i+1}) &= 6a_i h_i + 2b_i. && (4.7)
\end{aligned}$$

Gehen wir nun davon aus, dass auf dem ersten Intervall $[x_0, x_1]$ die Koeffizienten a_0, b_0, c_0, d_0 gegeben seien, so dass die Interpolationsbedingungen auf $[x_0, x_1]$ erfüllt sind. Mit anderen Worten, auf dem ersten Teilintervall wählen wir ein Polynom vom Grad $k = 3$, welches durch $k+1 = 4$ Freiheitsgrade beschrieben wird. Durch die Interpolationsbedingungen und die Stetigkeit von s' und s'' an den Knoten folgt:

$$\begin{aligned}
s_1(x_1) &= d_1 = f(x_1), && (4.8)\\
s_1(x_2) &= a_1 h_1^3 + b_1 h_1^2 + c_1 h_1 + d_1 = f(x_2), && (4.9)\\
s_1'(x_1) &= c_1 = 3a_0 h_0^2 + 2b_0 h_0 + c_0, && (4.10)\\
s_1''(x_1) &= 2b_1 = 6a_0 h_0 + 2b_0. && (4.11)
\end{aligned}$$

Aus den Gleichungen $(4.8) - (4.11)$ folgt die Eindeutigkeit der a_1, b_1, c_1, d_1. Hierbei sind b_1, c_1, d_1 durch die Stetigkeits- und Differenzierbarkeitsbedingungen $s \in C^2[a,b]$ am Knoten x_1 bestimmt. Die Interpolationsbedingung am Knoten x_2 legt den weiteren freien Parameter fest. Per Induktion folgt dann auch die eindeutige Bestimmung der weiteren a_k, b_k, c_k, d_k. In jedem weiteren Intervall haben wir also einen zusätzlichen Freiheitsgrad, welcher durch die Interpolationsbedingung festgelegt wird. D.h. der Raum $\mathcal{S}^3(\mathcal{T})$ mit $\mathcal{T} = \{x_0, \ldots, x_n\}$ besitzt $k+1+n-1 = n+3$ Freiheitsgrade. Allgemein läßt sich somit zeigen:

$$dim\, \mathcal{S}^k(\mathcal{T}) = n + k.$$

Untersuchen wir nun die Eigenschaften der kubischen Splines.

Satz 4.1.3 *Sei s ein interpolierender kubischer Spline zu der Funktion f an den Knoten $a = x_0 < \cdots < x_n = b$ und y eine beliebige interpolierende Funktion von f, sodass*

$$s''(x) \cdot \big(y'(x) - s'(x)\big) = 0, \quad x \in [a,b]. \tag{4.12}$$

Dann gilt

$$\|s''\|_2 := \Big(\int_a^b \big(s''(t)\big)^2\, dt \Big)^{1/2} \le \|y''\|_2. \tag{4.13}$$

Beweis. Aus (4.13) folgt mit $y'' = s'' + (y'' - s'')$

$$\begin{aligned}\int_a^b \big(y''(x)\big)^2\,dx &= \int_a^b \big(s''(x)\big)^2 + 2s''(x)\big(y''(x)-s''(x)\big) + \big(y''(x)-s''(x)\big)^2 dx \\ &= \int_a^b \big(s''(x)\big)^2 + \big(y''(x)-s''(x)\big)^2 dx \geq \int_a^b \big(s''(x)\big)^2 dx,\end{aligned}$$

falls $\int_a^b s''(y''-s'')dx$ verschwindet. Dies läßt sich aber mit (4.12) und partieller Integration unter Berücksichtigung von $s(x)\big|_{[x_{i-1},x_i]} \in \mathbb{P}_3$ (und somit $s'''(x)\big|_{[x_{i-1},x_i]} \equiv \text{const}_i =: C_i \in \mathbb{R}$) wiefolgt zeigen:

$$\begin{aligned}&\int_a^b s''(x)\big(y''(x)-s''(x)\big)dx = \sum_{i=1}^n \int_{x_{i-1}}^{x_i} s''(x)\big(y''(x)-s''(x)\big)dx \\ =\ & \sum_{i=1}^n \Big(\underbrace{s''(x)\big(y'(x)-s'(x)\big)}_{=0}\Big|_{x=x_{i-1}}^{x_i} - \int_{x_{i-1}}^{x_i} s'''(x)\big(y'(x)-s'(x)\big)dx\Big) \\ =\ & -\sum_{i=1}^n c_i \int_{x_{i-1}}^{x_i} y'(x)-s'(x)dx = -\sum_{i=1}^n c_i\Big[\big(y(x_i)-s(x_i)\big)-\big(y(x_{i-1})-s(x_{i-1})\big)\Big] = 0\end{aligned}$$

□

Die Aussage dieses Satzes benötigen wir insbesondere zum Beweis des folgenden wichtigen Resultats über Existenz und Eindeutigkeit eines interpolierenden kubischen Splines und dessen Minimaleigenschaften.

Satz 4.1.4 (Existenz, Eindeutigkeit und Minimaleigenschaft der kubischen Splines)
*Es sei $\mathcal{T} = \{x_0, \ldots, x_n\}$ eine Knotenfolge mit $a = x_0 < \cdots < x_n = b$ und $s \in \mathcal{S}^3(\mathcal{T})$ ein kubischer Spline, der neben den Interpolationsbedingungen $s(x_i) = f(x_i)$ eine der folgenden **Randbedingungen** erfülle:*

(i) $s'(a) = f'(a)$ *und* $s'(b) = f'(b)$ ***(vollständige Randbedingung)***

(ii) $s''(a) = s''(b) = 0$ ***(natürliche Randbedingung)***

(iii) $s'(a) = s'(b)$ *und* $s''(a) = s''(b)$ ***(periodische Randbedingung)***
(falls f periodisch mit Periode $b-a$ ist)

Ein solches $s \in \mathcal{S}^3(\mathcal{T})$ existiert und ist eindeutig bestimmt. Für jede interpolierende Funktion $y \in C^2[a,b]$, welche dieselben Interpolations- und Randbedingungen erfüllt, gilt ferner

$$\int_a^b \big(s''(x)\big)^2\,dx \leq \int_a^b \big(y''(x)\big)^2\,dx.$$

Beweis. Die Interpolations- und Randbedingungen sind linear in s, und ihre Anzahl stimmt mit der Dimension von $\mathcal{S}^k(\mathcal{T})$ überein. Somit genügt es zu zeigen, dass für die $f \equiv 0$ der triviale Spline $s \equiv 0$ die einzige Lösung ist.

Da $y \equiv 0$ alle Bedingungen erfüllt, folgt mit Satz 4.1.3, dass auch $\|s''\| = 0$ gilt. Da s'' stetig, folgt somit auch $s'' = 0$, somit $s' = c_0$ und $s(x) = c_0 x + c_1$. Aus den Interpolationsbedingungen $s(x_i) = 0$ folgt sofort $s = 0$. Diese Lösung ist nach obrigem eindeutig. □

Bemerkung 4.1.5 *Der Satz 4.1.4 gilt unter Verallgemeinerung der Randbedingungen für beliebige Splineräume $\mathcal{S}^k(\mathcal{T})$, $k \geq 3$. Siehe z.B. [Hämmerlin/Hoffmann], Seite 252.*

4.1.1 Berechnung kubischer Splines

Kommen wir nun zur Berechnung kubischer Splines. Zusätzlich zu den Daten

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

ist es zweckmäßig, die Krümmungen als Variablen

$$y_i'' := s''(x_i), \quad i = 0, \ldots, n-1,$$

einzuführen und die $4n$ Variablen $a_i, b_i, c_i, d_i, i = 0, \ldots, n-1$, d.h. die Koeffizienten der stückweisen Polynome $s_i, i = 0, \ldots, n-1$ vom Grad 3 durch diese $n+1$ Variablen zu ersetzen. Aus den Gleichungen $(4.2), (4.3)$ sowie $(4.6), (4.7)$ und mit den Krümmungen an den inneren Knoten $y_i'' = s''(x_i) = s_i''(x_i) (= s_{i+1}''(x_i))$, $i = 1, \ldots, n-1$ sowie den Krümmungen an den äußeren Knoten $y_0'' = s''(x_0) = s_i''(x_0)$, $y_n'' = s''(x_n) = s_{n-1}''(x_n)$ gewinnt man das Gleichungssystem

$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ h_i^3 & h_i^2 & h_i & 1 \\ 0 & 2 & 0 & 0 \\ 6h_i & 2 & 0 & 0 \end{pmatrix} \begin{pmatrix} a_i \\ b_i \\ c_i \\ d_i \end{pmatrix} = \begin{pmatrix} s_i(x_i) \\ s_i(x_{i+1}) \\ s_i''(x_i) \\ s_i''(x_{i+1}) \end{pmatrix} = \begin{pmatrix} y_i \\ y_{i+1} \\ y_i'' \\ y_{i+1}'' \end{pmatrix}. \tag{4.14}$$

Die Berechnung der Determinanten der Koeffizientenmatrix liefert

$$\det \begin{pmatrix} 0 & 0 & 0 & 1 \\ h_i^3 & h_i^2 & h_i & 1 \\ 0 & 2 & 0 & 0 \\ 6h_i & 2 & 0 & 0 \end{pmatrix} = -\det \begin{pmatrix} h_i^3 & h_i^2 & h_i \\ 0 & 2 & 0 \\ 6h_i & 2 & 0 \end{pmatrix} = -2 \det \begin{pmatrix} h_i^3 & h_i \\ 6h_i & 0 \end{pmatrix} = 12h_i^2$$

und damit die Eindeutigkeit der a_i, b_i, c_i, d_i, also von s, falls s_i und s_i'' für alle $i = 0, \ldots, n-1$, bekannt sein sollten. Das Lösen von (4.14) liefert nun für $i = 0, \ldots, n-1$,

$$d_i = y_i \qquad b_i = \frac{1}{2} y_i'' \tag{4.15}$$

$$a_i = \frac{1}{6h_i}(y_{i+1}'' - y_i'') \qquad c_i = \frac{1}{h_i}(y_{i+1} - y_i) - \frac{1}{6} h_i (y_{i+1}'' + 2y_i''). \tag{4.16}$$

Es lassen sich somit die kubischen Polynome $s_i(x)$ in jedem Teilintervall eindeutig bestimmen, wenn neben den Stützwerten y_k auch die Größen y_k'' bekannt sind. Damit wäre neben der Interpolationseigenschaft auch gleich die Stetigkeit der zweiten Ableitung von $s(x)$ gesichert.

Was nun zu zeigen wäre, ist, dass aus den y_i, y_i'' auch die Stetigkeit von $s(x)$ folgt. Setzen wir die Darstellung der a_i, b_i, c_i und d_i in (4.5) ein, so erhalten wir für $i = 0, \ldots, n-2$,

$$\begin{aligned} s_i'(x_{i+1}) &= 3h_i^2 a_i + 2h_i b_i + c_i \\ &= 3h_i^2 \left(\frac{1}{6h_i}(y_{i+1}'' - y_i'') \right) + 2h_i \left(\frac{1}{2} y_i'' \right) + \frac{1}{h_i}(y_{i+1} - y_i) - \frac{1}{h_i}(y_{i+1}'' + 2y_i'') \\ &= h_i \left(\frac{1}{2} y_{i+1}'' - \frac{1}{2} y_i'' + y_i'' - \frac{1}{6} y_{i+1}'' - \frac{1}{3} y_i'' \right) + \frac{1}{h_i}(y_{i+1} - y_i) \\ &= \frac{h_i}{6}(2y_{i+1}'' + y_i'') + \frac{1}{h_i}(y_{i+1} - y_i), \end{aligned}$$

bzw.

$$s_{i-1}'(x_i) = \frac{1}{h_{i-1}}(y_i - y_{i-1}) + \frac{h_{i-1}}{6}(2y_i'' + y_{i-1}''), \quad i = 1, \ldots, n-1. \tag{4.17}$$

Die Stetigkeit von $s'(x)$ an den inneren Knoten liefert mit der letzten Gleichung (4.17) wegen der Darstellung von c_i aus (4.4) und (4.16) die Gleichung

$$\begin{aligned} & \frac{1}{h_{i-1}}(y_i - y_{i-1}) + \frac{1}{6}h_{i-1}(2y_i'' + y_{i-1}'') \\ = \quad & s_{i-1}'(x_i) \stackrel{!}{=} s_i'(x_i) = c_i = \frac{1}{h_i}(y_{i+1} - y_i) - \frac{1}{6}h_i(y_{i+1}'' + 2y_i''). \end{aligned}$$

Sortieren von y_k'' nach links, y_k nach rechts und anschließende Multiplikation mit 6 liefert

$$h_{i-1}y_{i-1}'' + 2(h_{i-1} + h_i)y_i'' + h_i y_{i+1}'' = \frac{6}{h_i}(y_{i+1} - y_i) - \frac{6}{h_{i-1}}(y_i - y_{i-1}). \tag{4.18}$$

Diese Bedingung muss für alle inneren Knoten $x_1, \ldots, x_{n-1}$ erfüllt sein und liefert somit $n-1$ Gleichungen für die $n+1$ Unbekannten $y_0'', \ldots, y_n''$.

Allerdings reichen die $n-1$ Gleichungen für $n+1$ Unbekannte $y_0'', \ldots, y_n''$ nicht aus, um diese eindeutig zu bestimmen. Untersuchen wir nun die unterschiedliche Behandlung der Randbedingungen.

4.1.1.1 Berücksichtigung natürlicher und vollständiger Randbedingungen

Die **vollständige Randbedingung** $s'(a) = f'(a)$, bzw. $s'(b) = f'(b)$ liefert aufgrund von (4.4), (4.5) und (4.16) die weitere Gleichung

$$s'(a) = s_0'(x_0) = c_0 = \frac{1}{h_0}(y_1 - y_0) - \frac{1}{6}(y_1'' + 2y_0'') \stackrel{!}{=} f'(a)$$

für den linken Rand. Somit folgt

$$2h_0y_0'' + h_0y_1'' = \frac{6}{h_0}(y_1 - y_0) - 6f'(a), \tag{4.19}$$

bzw. aus

$$\begin{aligned} s'(b) &= s_{n-1}'(x_n) = 3a_{n-1}h_{n-1}^2 - 2b_{n-1}h_{n-1} + c_{n-1} \\ &= \frac{h_{n-1}}{2}(y_n'' - y_{n-1}'') + h_{n-1}y_{n-1}'' + \frac{1}{h_{n-1}}(y_n - y_{n-1}) - \frac{1}{6}h_{n-1}(y_n'' + 2y_{n-1}'') \\ &= h_{n-1}\left(\frac{1}{2}y_n'' - \frac{1}{2}y_{n-1}'' + y_{n-1}'' - \frac{1}{6}y_n'' - \frac{1}{3}y_{n-1}''\right) + \frac{1}{h_{n-1}}(y_n - y_{n-1}) \\ &= h_{n-1}\left(\frac{1}{3}y_n'' + \frac{1}{6}y_{n-1}\right) + \frac{1}{h_{n-1}}(y_n - y_{n-1}) \stackrel{!}{=} f'(b), \end{aligned}$$

für den rechten Rand

$$h_{n-1}y_{n-1}'' + 2h_{n-1}y_n'' = -\frac{6}{h_{n-1}}(y_n - y_{n-1}) + 6f'(b). \tag{4.20}$$

Für die **natürlichen Randbedingung** $y_0'' = y_n'' = 0$ ergeben sich folgende Gleichungen

$$y_0'' = 0, \tag{4.21}$$
$$y_n'' = 0. \tag{4.22}$$

Natürliche und vollständige Randbedingungen können auch gemischt auftreten, d.h. an der Stelle x_0 ist neben $f(x_0)$ auch die Ableitung $f'(x_0)$ vorgegeben und an der Stelle x_n gilt $y_n = 0$.
Im Falle $n = 5$ erhalten wir daraus folgendes lineares Gleichungssystem:

$$\begin{pmatrix} \star & \star & \star & \star & \star & \star \\ h_0 & 2(h_0+h_1) & h_1 & & & \\ & h_1 & 2(h_1+h_2) & h_2 & & \\ & & h_2 & 2(h_2+h_3) & h_3 & \\ & & & h_3 & 2(h_3+h_4) & h_4 \\ \star & \star & \star & \star & \star & \star \end{pmatrix} \cdot \begin{pmatrix} y_0'' \\ \vdots \\ \\ \vdots \\ y_5'' \end{pmatrix}$$

$$= \begin{pmatrix} & \star & \\ \frac{6}{h_1}(y_2-y_1) - \frac{6}{h_0}(y_1-y_0) \\ \frac{6}{h_2}(y_3-y_2) - \frac{6}{h_1}(y_2-y_1) \\ \frac{6}{h_3}(y_4-y_3) - \frac{6}{h_2}(y_3-y_2) \\ \frac{6}{h_4}(y_5-y_4) - \frac{6}{h_3}(y_4-y_3) \\ & \star & \end{pmatrix}. \tag{4.23}$$

Die in diesem Schema mittels $\star$ gekennzeichnete erste und letzte Zeile ist dabei durch die Wahl der Randbedingungen (4.19), (3.22) bzw. $(3.35), (3.36)$ gegeben.
Für vollständige Randbedingungen sind in (4.23) erste und letzte Zeile durch die Gleichungen (4.19) und (4.21) zu ersetzen, sodass wir erhalten:

$$\begin{pmatrix} 2h_0 & h_0 & & & \\ \star & \star & \star & & \\ & & \ddots & & \\ & & \star & \star & \star \\ & & & h_{n-1} & 2h_{n-1} \end{pmatrix} \begin{pmatrix} y_0'' \\ \star \\ \vdots \\ \star \\ y_n'' \end{pmatrix} = \begin{pmatrix} \frac{6}{h_0}(y_1-y_0) - 6f'(a) \\ \star \\ \vdots \\ \star \\ -\frac{6}{h_{n-1}}(y_n-y_{n-1}) + 6f'(b) \end{pmatrix}.$$

Hierbei sind die mit $\star$ gekennzeichneten Zeilen 1 bis $n-1$ dem System (4.23) zu entnehmen.
Analog erhalten wir für die Randbedingungen $y_0'' = f''(a)$, $y_n'' = f''(b)$

$$\begin{pmatrix} 1 & & & & \\ \star & \star & \star & & \\ & & \ddots & & \\ & & \star & \star & \star \\ & & & & 1 \end{pmatrix} \begin{pmatrix} y_0'' \\ \star \\ \vdots \\ \star \\ y_n'' \end{pmatrix} = \begin{pmatrix} f''(a) \\ \star \\ \vdots \\ \star \\ f''(b) \end{pmatrix}.$$

Da im letzten Gleichungssystem y_0'' und y_n'' explizit bekannt sind, können wir es wie folgt reduzieren:

$$\begin{pmatrix} 2(h_0+h_1) & h_1 & & & \\ h_1 & 2(h_1+h_2) & h_2 & & \\ & & \ddots & & \\ & & h_{n-2} & 2(h_{n-2}+h_{n-1}) \end{pmatrix} \cdot \begin{pmatrix} y_1'' \\ \vdots \\ \vdots \\ y_{n-1}'' \end{pmatrix} = \begin{pmatrix} \star & -h_0 y_0'' \\ \star & \\ \star & \\ \star & -h_{n-1} y_n'' \end{pmatrix}$$

Bemerkung 4.1.6 *Man beachte, dass $y_0'' = C_0$ und $y_n'' = C_1$ mit $C_0, C_1 \in \mathbb{R}$ nicht die Matrix, sondern nur die rechte Seite modifiziert, damit also auch eine Verallgemeinerung der natürlichen Randbedingungen möglich ist.*

4.1.1.2 Berücksichtigung periodischer Randbedingungen

Die Stützstellen $x_0 < \cdots < x_n$ seien nun so festgelegt, dass $x_n = x_0 + T$, wobei T die Periode der gesuchten Funktion darstelle.

Die periodischen Randbedingungen sind

$$f(x_0) = f(x_n), \quad f'(x_0) = f'(x_n) \quad \text{sowie} \quad f''(x_0) = f''(x_n).$$

Mit den Größen $y_0 = y_n$, $y_1, \ldots, y_n$ und den Unbekannten $y_0'' = y_n'', y_1'', \ldots, y_{n-1}''$ ist die Stetigkeit an den n Stützstellen $x_0, \ldots, x_{n-1}$ zu erfüllen (beachte, dass die Stetigkeit bei x_0 der bei x_n aufgrund der Periodizität entspricht).

Für die inneren Knoten $x_1, \ldots, x_{n-1}$ sind die Gleichungen gegeben durch (4.18). Aber auch zum Knoten x_0 sind die Gleichungen durch (4.18) gegeben, wenn man bei der Formulierung

$$\begin{aligned} h_{-1} &= x_0 - x_{-1} = x_n - x_{n-1} = h_{n-1} \\ y_{-1}'' &= y_{n-1}'' \text{ und } y_{-1} = y_{n-1} \end{aligned}$$

beachtet, da zu jedem Knoten x_i nur die Informationen y_{i-1}, y_i, y_{i+1} und $y_{i-1}'', y_i'', y_{i+1}''$ benötigt werden, d.h. die Daten vom linken, rechten Nachbarn sowie vom Knoten selbst.

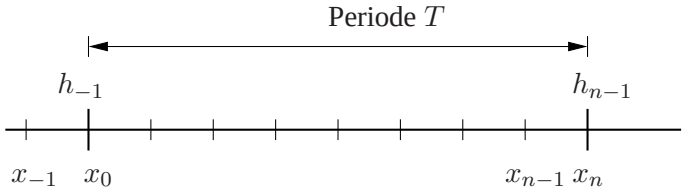


Abb. 4.3: Nummerierung im Fall periodischer Randbedingung mit Periode T

Das System lautet dann allgemein für $n = N, x_{N+1} = x_0 + T$

$$\begin{pmatrix} 2(h_N + h_0) & h_0 & & & & & h_N \\ h_0 & 2(h_0 + h_1) & h_1 & & & & \\ & h_1 & 2(h_1 + h_2) & h_2 & & & \\ & & \ddots & \ddots & & & \\ & & & h_{N-2} & 2(h_{N-2} + h_{N-1}) & h_{N-1} & \\ h_N & & & & h_{N-1} & 2(h_{N-1} + h_N) & \end{pmatrix} \cdot \begin{pmatrix} y_0'' \\ y_1'' \\ y_2'' \\ \vdots \\ \vdots \\ y_{N-1}'' \\ y_N'' \end{pmatrix}$$

$$= \begin{pmatrix} \frac{6}{h_0}(y_1 - y_0) - \frac{6}{h_N}(y_0 - y_N) \\ \frac{6}{h_1}(y_2 - y_1) - \frac{6}{h_0}(y_1 - y_0) \\ \frac{6}{h_2}(y_3 - y_2) - \frac{6}{h_1}(y_2 - y_1) \\ \vdots \\ \frac{6}{h_{N-1}}(y_N - y_{N-1}) - \frac{6}{h_{N-2}}(y_{N-1} - y_{N-2}) \\ \frac{6}{h_N}(y_0 - y_N) - \frac{6}{h_{N-1}}(y_N - y_{N-1}) \end{pmatrix}$$

Bemerkung 4.1.7 *Die natürlichen Randbedingungen entsprechen also gerade der Situation, dass das Zeichenwerkzeug außerhalb des Interpolationsintervalls gerade ist. Da bei den vollständigen Randbedingungen mehr Informationen über die zu interpolierende Funktion eingehen, sind auch deren Interpolationseigenschaften besser.*

Eine Matlab-Realisierung zur Berechnung der Koeffizienten a_i, b_i, c_i, d_i für alle Intervalls $[x_i, x_{i+1}]$, $i = 0, \ldots, n-1$) ist im Folgenden wieder gegeben.

MATLAB-Funktion: CoeffSpline.m

```
function coeff = CoeffSpline(t,y,kind,param)
% param(1,1)=f'(a) bzw. f''(a) (abhängig von der Wahl von kind)
% param(2,1)=f'(b) bzw. f''(b)
n=length(t);
dy = y(2:end)-y(1:end-1);
%% Berechnung des "Kerns" von A und b
h1 = t(2:n)-t(1:n-1);
h2 = t(3:end)-t(1:end-2);
A = sparse(n,n);
B = [h1,[2*h2;0],[h1(2:end);0]];
A(2:n-1,:)=spdiags(B,[0,1,2],n-2,n);
b = zeros(n,1);
b(2:n-1)=6*((y(3:n)-y(2:n-1))./h1(2:n-1)-...
         (y(2:n-1)-y(1:n-2))./h1(1:n-2));
%% Erweiterung von A und b für unterschiedliche Randbedingungen
switch kind
  case 'nat'
    A(1,1)=1; A(n,n)=1;
    b(1)=param(1); b(end)=param(2);
  case 'per'
    if y(1)~=y(end)
      error('This data is not applicable for periodic splines')
    end
    A(1,[1,2,n-1])=[2*(h1(1)+h1(end)),h1(1),h1(end)];
    A(n,[1,n])=[1,-1];
    b(1)=6*((y(2)-y(1))/h1(1)-(y(1)-y(end-1))/h1(end));
  case 'compl'
    A(1,[1,2])=[2*h1(1),h1(1)];
    A(n,[n-1,n])=[h1(end),2*h1(end)];
    b(1)=6*(y(2)-y(1)-param(1));
    b(n)=-6*(y(end)-y(end-1)+param(2));
  otherwise
    error('This kind does not exist!')
end
%% Berechnung der y''
y2d=A\b;
%% Berechnung der Koeffizienten a,b,c,d
coeff=[y(1:end-1),...
        (y(2:end)-y(1:end-1))./h1(1:end)-h1(1:end).*(y2d(2:end)+2*
           y2d(1:end-1))/6,...
        y2d(1:end-1)/2,(y2d(2:end)-y2d(1:end-1))./(6*h1(1:end))]';
```

4.1.2 Punktauswertung kubischer Splines

Nachdem wir im letzten Abschnitt analysiert haben, wie sich aus den Daten $a = x_0 < x_1 < \cdots < x_n = b$ und $y_0, \ldots, y_n$ ein kubischer Spline bestimmen läßt, stellt sich nun die Frage, wie wir diesen auswerten können. Im Gegensatz zu einem Polynom, sind die Splines nur jeweils stückweise definiert, d.h. wollen wir an einer Stelle $x \in [a, b]$ den Spline $s(x)$ auswerten, müssen wir erst j mit $x \in [x_j, x_{j+1}]$ bestimmen.

4.1.2.1 Sequentielle Suche

Bei der trivialen Realisierung, bei der man nacheinander $j = 0, 1, \ldots, n-1$ durchgeht und testet, ob x in $[x_j, x_{j+1}]$ liegt, benötigt man bei einer äquidistanten Unterteilung $|x_{j+1} - x_j| =: h$, $j = 0, \ldots, n-1$ in n Teilintervalle durchschnittlich $(n-1)/2$ Abfragen. Da x mit der gleichen Wahrscheinlichkeit $1/n$ in einem der Intervalle $[x_j, x_{j+1}]$, $j = 0, \ldots, n-1$, liegt, wird bei einem x, welches im letzten $(n$-ten$)$ Intervall $[x_{n-1}, x_n]$ oder vorletzten Intervall $[x_{n-2}, x_{n-1}]$ vorher $(n-1)$-mal ein Test auf „x liegt im Intervall“ durchgeführt. Für ein x, welches im j-ten Intervall $[x_{j-1}, x_j]$ liegt, wird eine solche Überprüfung j-mal durchgeführt, d.h. durchschnittlich werden

$$\frac{1}{n} + \frac{2}{n} + \ldots + \frac{n-2}{n} + \frac{n-1}{n} + \frac{n-1}{n} = \frac{n(n-1)}{2n} + \frac{n-1}{n} \approx \frac{n-1}{2} + 1$$

Tests benötigt.

4.1.2.2 Binäre Suche

Wie man schnell sieht, ist man mit einer binären Suche deutlich schneller. Vereinfachen wir die Voraussetzungen insofern, dass wir von $n = 2^s$ Intervallen ausgehen und unser x liege mit der gleichen Wahrscheinlichkeit in einem der Teilintervalle. Mit einem Test ob x in den ersten 2^{s-1} oder den letzten 2^{s-1} Intervallen liegt, reduzieren wir die Problemgröße auf die Hälfte und erhalten nach s Tests das gesuchte Intervall.

Gilt nun $2^{s-1} < n \leq 2^s$, so wähle man $2^s - n$ virtuelle Intervalle $[a, b]$ und nach s Bisektionen hat man das gesuchte Intervall gefunden. Die Anzahl der Tests ist für $2^{s-1} < n \leq 2^s$ gerade $s = \lceil \log_2 n \rceil$.

Beispiel 4.1.8 (Vergleich seuentielle und binäre Suche) Für $n = 100(10000)$ benötigt man durchschnittlich bei der sequentiellen Suche dem $51(5001)$ Tests und bei der binären Suche nur $7(14)$ Tests.

Nachfolgend führen wir die entsprechenden *Matlab*-Zeilen zur Auswertung mittels sequentieller und binärer Suche an. In beiden Fällen wird das Vorgehen durch die entsprechende *Matlab*-Ausgabe verdeutlicht:

MATLAB-Funktion: sequentiellesuche.m

```
function k = sequentielle_suche(x,x0)
% x_1 < x_2 < x_3 <  < x_n
% Finde kleinstes k, sodass  x0 in [x_k,x_(k+1)]
% und setze k = 0, wenn es kein solches k gibt
for k = 1:length(x)-1
    if x(k) <= x0 && x0 <= x(k+1)
      return
    end
end
k = 0;
```

MATLAB-Funktion: binaeresuche.m

```
function k_unten = binaeresuche(x,x0)
% x_1 < x_2 < x_3 <   < x_n
% Finde kleinstes k so dass x0 in [x_k,x_(k+1)]
% und setze k = 0, wenn es kein solches k gibt
if x0 < x(1) || x(end) < x0
    k_unten = 0;
    return
end
k_unten = 1;
k_oben = length(x);
while k_oben - k_unten > 1
    k_mitte = floor((k_unten + k_oben)/2);   %rundet -> -inf
    if x(k_unten)<= x0 && x0 <= x(k_mitte)
         k_oben = k_mitte;
    else
         k_unten = k_mitte;
    end
end
```

MATLAB-Beispiel:

Als Ausgabe erhalten wir:

```
>> N=10^6,x=[1:N];x0=N*rand(100,1);
>> tic, for k=1:100,
   sequentiellesuche(x,x0(k)); end, toc
N =
     1000000
Elapsed time is 1.844000 seconds.
>> N=10^6,x=[1:N];x0=N*rand(100,1);
>> tic, for k=1:100,
   binaeresuche(x,x0(k)); end, toc
N =
     1000000
Elapsed time is 0.016000 seconds.
```

4.1.2.3 Suche mit korrelierten Daten

Häufig kommt man bei der Auswertung des Splines noch zu einer speziellen Situation, nämlich die Auswertung von s an einer aufsteigenden Folge $t_j \leq t_{j+1}$ von Punkten $t_j \in [a,b]$, $j = 1,\ldots,m$. Gilt $t_j \in [x_{k_j}, x_{k_j+1}]$, $k_j \in \{0,\ldots,n-1\}$, so ist klar, dass man t_{j+1} nur noch in der Teilmenge $[x_{k_j}, b]$ suchen muss. Wäre nur noch ein Intervall zu suchen, böte die Bisektionsmethode einen effizienten Suchalgorithmus, wenn aber noch mehrere Intervalle für $t_{j+1},\ldots,t_m$ zu lokalisieren sind, wird das nächste k_{j+1} in der Nähe des zuletzt bestimmten k_j liegen. Dies muss aber keineswegs dasselbe oder auch das nächste Intervall sein. Die Idee des folgenden „Jagd“-Algorithmus ist es, das nächste k_{j+1} durch *größer* werdende Schritte einzuschachteln.

Gilt $t_{j+1} \notin [x_{k_j}, x_{k_j+1}]$ so teste man nacheinander

$$\begin{aligned} t_{j+1} &\in [x_{k_j+1}, x_{k_j+2}] \,? \\ t_{j+1} &\in [x_{k_j+2}, x_{k_j+4}] \,? \\ t_{j+1} &\in [x_{k_j+4}, x_{k_j+8}] \,? \\ &\vdots \end{aligned}$$

Hat man hierdurch ein Intervall identifiziert, wendet man bezüglich diesem Intervall die Bisektionsmethode an. Im „Worstcase“ benötigt man zweimal länger als mit der Bisektionssuche, aber im besten Fall ist man um den Faktor $\log_2 n$ schneller.

Nachfolgend der oben erwähnte „Jagd“-Algorithmus in *MATLAB*-Implementierung:

MATLAB-Funktion: lokalisierejagd.m

```
function ks = lokalisierejagd(xs,x0)
% xs(1) < xs(2) < xs(3) < ... < xs(n)
% Finde für alle x0's die kleinsten k's, sodass x0 in [xs(k),xs(k
    +1)]
  ks = zeros(length(x0),1);
  n = length(xs);
  k = 1;
  for j = 1:length(x0)
    inc = 1;
    k_next = k + 1;
    while k_next <= n && x0(j) > xs(k_next) % hunting
      k = k_next;
      k_next = k_next + inc;
      inc = 2 * inc;
    end
    k_next = min(k_next,n);
    if k_next > k+1                          % bisection
      k = k + binaeresuche(xs(k:k_next),x0(j)) - 1;
    end
    ks(j) = k;
  end
end
```

Besteht der Spline aus n Intervallen und gesucht ist die Auswertung an $m \ll n$ Stützstellen so benötigen `lokalisierejagd` und `binaeresuche` etwa gleich viel Zeit.

MATLAB-Beispiel:

```
>> m = 20001; n = 80001;
>> nodes = linspace(0,1,n);
>> x0 = linspace(0,1,m);
>> tic, s = lokalisierejagd(nodes,x0);toc
Elapsed time is 0.448687 seconds.
>> t=zeros(m,1);
>> tic,for j = 1:m,t(j) = binaeresuche(
   nodes,x0(j)); end,toc
Elapsed time is 0.349985 seconds.
```

Ist jedoch die Anzahl der Auswertung deutlich größer als die Anzahl der Intervalle, auf dem der Spline stückweise definiert ist, so ist `lokalisierejagd` deutlich schneller als `binaeresuche`.

```
>> m = 80001; n = 20001;
>> nodes = linspace(0,1,n);
>> x0 = linspace(0,1,m);
Elapsed time is 0.023416 seconds.
>> tic, s = lokalisierejagd(nodes,x0);toc
>> t=zeros(m,1);
>> tic,for j = 1:m,t(j) = binaeresuche(
   nodes,x0(j)); end,toc
Elapsed time is 1.382369 seconds.
```

4.2 *Bézier*-TECHNIK

Computer-gestützte Konstruktionen von Objekten, wie z.B. durch CAD-Anwendungen (Computer Aided Design), erfordern das schnelle Zeichnen und Manipulieren von Kurven und Flächen. In den sechziger Jahren entwickelten, *Bézier*[1] und *de Casteljau*[2], die als Ingenieure bei den französischen Automobilherstellern Renault und Citroen beschfäftigt waren, die Idee für ein Verfahren zur Umwandlung einer beliebigen Form in eine mathematische Vorschrift einer Kurve.

Als Vorbereitung für die beiden folgenden Abschnitte werden wir hier zunächst auf die mathematische Beschreibung und Darstellung von Kurven und Flächen eingehen.

4.2.1 Parametrisierte Kurven und Flächen

Die beiden häufigsten Methoden, um Kurven oder Flächen mathematisch zu beschreiben, sind die implizite Darstellung und die parametrisierte Form.

Die **implizite Darstellung** einer Kurve in der xy-Ebene hat die Form

$$f(x, y) = 0.$$

Zu einer gegeben Kurve ist diese Darstellung eindeutig bis auf eine multiplikative Konstante. Ein Beispiel ist der Einheitskreis, definiert durch die Gleichung $f(x, y) = x^2 + y^2 - 1 = 0$.

In der **Parameterdarstellung** wird jede Koordinate eines Punkts auf der Kurve seperat durch eine explizite Funktion eines unabhängigen Parameters dargestellt,

$$\mathbf{C}(t) = (x(t), y(t))^T, \quad a \le t \le b\,.$$

Somit ist $\mathbf{C}(t)$ eine vektorwertige Funktion des Parameteres t. Obwohl das Intervall $[a, b]$ beliebig sein kann, wird es üblicherweise auf $[0, 1]$ normiert. Der erste Quadrant des Einheitskreises ist definiert durch die Paremeterdarstellung

$$x(t) = \cos(t), \quad y(t) = \sin(t), \quad a \le t \le \pi/2\,.$$

Substituiert man $u = \tan(t/2)$ so erhällt man die alternative Darstellung

$$x(u) = \frac{1 - u^2}{1 + u^2}, \quad y(u) = \frac{2u}{1 + u^2}, \quad 0 \le u \le 1\,.$$

Die parametrische Darstellung ist folglich nicht eindeutig.

Beide Darstellungsformen haben **Vor- und Nachteile**, von denen einige hier genannt seien.

[1] Bézier, Pierre (1910-1999)
[2] de Casteljau, Paul (1930-)

- Fügt man eine z-Koordinate hinzu, so lässt sich die gegebene Parameterdarstellung einer Kurve einfach in 3-dimensionalen Raum einbetten. Durch die implizite Form lassen sich nur Kurven in der xy- (oder yz- oder yz-) Ebene darstellen.
- Parametrisierte Kurven haben eine natürliche Richtung (von $\mathbf{C}(a)$ zu $\mathbf{C}(b)$ für $a \leq t \leq b$). Somit lassen sich einfach geordnete Folgen von Punkten erzeugen. Implizit gegebene Kurven haben diese Eigenschaft nicht.
- In der Parameterdarstellung muss man manchmal mit „Anomalien kämpfen“, die nicht im Zusammenhang stehen mit der wirklichen Geometrie. Ein Beispiel ist die Einheitskugel. Verwendet man Kugelkoordianten so sind die Pole algorithmisch schwierige Punkte, obwohl sie sich von den anderen Punkten nicht unterscheiden.
- Die Komplexität vieler geometrischer Operationen und Manipulationen hängt stark von der Darstellung ab. Die Berechnung eines Punktes auf einer Kurve ist schwierig in der impliziten Darstellung. Die Entscheidung, ob ein Punkt auf einer Kurve oder Fläche liegt ist jedoch in impliziten Darstellung einfacher.
- Unbeschränkte Geometrien lassen sich nur schwer mit einer Parameterdarstellung beschreiben.

Lässt man beliebige Koordinatenfunktionen $x(t)$, $y(t)$, $z(t)$ zur Beschreibung von Kurven zu, so erhällt man eine riesige Auswahl an möglichen Kurven. Möchte man dies aber mit Hilfe eines Rechners umsetzen, so gibt es einige Restriktionen zu berücksichtigen. Am besten wäre es, man beschränkt sich auf eine Klasse von Funktionen, die

- die gewünschten Kurven präzise genug darstellt, wie sie für Berechnungen oder Darstellungen benötigt werden,
- einfach, effizient und stabil sind,
- wenig Speicherplatz benötigen,
- mathematisch einfach gut verstanden sind (d.h. keine Heuristiken).

Eine naheliegende Wahl von Funktionen wären die Polynome. Obwohl sie die letzten beiden Punkte in der Wunschliste erfüllen, gibt es mehere wichtiger Kurven und Flächen, die sich nicht durch Polynome darstellen lassen, z.B. Kreise und Kugeln.

Die Darstellung einer Kurve in monomialer Basis n-ten Grades ist gegeben durch

$$\mathbf{C}(t) = (x(t), y(t), z(t))^T = \sum_{j=0}^{n} \mathbf{a}_j t^j, \quad 0 \leq t \leq 1,$$

mit $\mathbf{a}_j = (x_j, y_j, z_j)^T$. Zu einem gegebenem t_0 lässt sich der Punkt $\mathbf{C}(t_0)$ möglichst effizient mit dem **Horner-Schema** berechnen:

- für den Grad $= 1$: $\mathbf{C}(t_0) = \mathbf{a}_0 + t_0\mathbf{a}_1$
- Grad $= 2$: $\mathbf{C}(t_0) = \mathbf{a}_0 + t_0(\mathbf{a}_1 + t_0\mathbf{a}_2)$
- Grad $= 3$: $\mathbf{C}(t_0) = \mathbf{a}_0 + t_0(\mathbf{a}_1 + t_0(\mathbf{a}_2 + t_0\mathbf{a}_3))$
- $\vdots$
- Grad $= n$: $\mathbf{C}(t_0) = \mathbf{a}_0 + t_0(\ldots t_0(\mathbf{a}_{n-2} + \ldots t_0(\mathbf{a}_{n-1} + t_0\mathbf{a}_n)))$

In Matlab sieht der Algorithmus wie folgt aus.

MATLAB-Funktion: horner.m

```
function f = horner(a,t0)
% Compute point on a power basis curve
f = a(:,end);
for k = size(a,2)-1:-1:1
  f = f.*t0+a(:,k);
end
```

4.2.2 *Bernstein*-Polynome

Die monomile Basis ist nicht die einzige, um Polynome darzustellen. In Rahmen der Interpolation wurden auch schon die *Lagrange*- und *Newton*-Basis diskutiert. Wir definieren nun zuerst eine weitere Basis für $\mathbb{P}_n$, nämlich die ***Bernstein***[3]**-Polynome**. Obwohl die parametrisierten Funktionen, dargestellt in monomialer Basis oder mit *Bernstein*-Polynomen, mathematisch äquivalent sind, ist die Darstellung mit Hilfe der *Bernstein*-Polynome für die Darstellung von Kurven und Flächen deutlich geeigneter. An entsprechender Stelle kommen wir auf diesen Punkt zurück.

Definition 4.2.1 (*Bernstein*-Polynom) *Das i-te **Bernstein**-Polynom vom Grad n bezüglich des Intervalls $[0,1]$ ist das Polynom $B_i^n \in \mathbb{P}_n$ mit*

$$B_i^n(t) = \frac{n!}{i!(n-i)!} t^i (1-t)^{n-i}, \quad i = 0, \ldots, n. \tag{4.24}$$

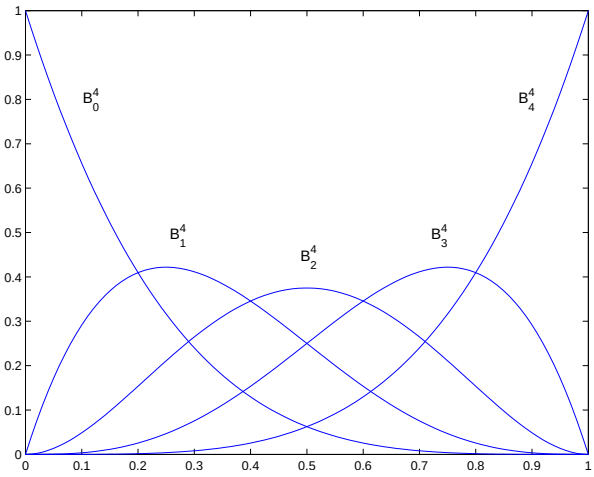


Abb. 4.4: Bernstein-Polynome $B_0^4, \ldots, B_0^4$ auf dem Intervall $[0,1]$.

Bemerkung 4.2.2 (Affine Transformation der *Bernstein*-Polynome) *Mit der affinen Transformation*

$$\lambda(t) \in [a,b] \to t \in [0,1], \quad \lambda(t) := \frac{t-a}{b-a}$$

[3] Bernstein, Sergei Natanowitsch (1880-1968)

kann man ein beliebiges Intervall $[a,b]$ *auf das Einheitsintervall* $[0,1]$ *transformieren. Das* i*-te* Bernstein*-Polynom vom Grad* n *bezüglich des Intervalls* $[a,b]$ *ist dann* $B_i^n(\cdot;a,b) \in \mathbb{P}_n$ *mit*

$$B_i^n(t;a,b) := B_i^n(\lambda(t)) = B_i^n\left(\frac{t-a}{b-a}\right) = \frac{1}{(b-a)^n}\begin{pmatrix} n \\ t \end{pmatrix}(t-a)^i(b-t)^{n-t}.$$

Satz 4.2.3 (Eigenschaften der *Bernstein*-Polynome) *Die* Bernstein*-Polynome haben folgende Eigenschaften:*

(i) ***Positivität:*** $B_i^n(t) \geq 0$ *für alle* i,n *und* $0 \leq t \leq 1$.

(ii) ***Zerlegung der Eins:*** $\sum_{i=0}^n B_i^n(t) = 1$ *für alle* $0 \leq t \leq 1$.

(iii) $B_0^n(0) = B_n^n(1) = 1$.

(iv) $B_i^n(t)$ *hat genau ein Maximum im Intervall* $[0,1]$*, und zwar bei* $t = i/n$.

(v) ***Symmetrie:*** $B_i^n(t) = B_{n-i}^n(1-t)$ *für* $i = 0,\ldots,n$.

(vi) ***Rekursionsformel:*** $B_i^n(t) = (1-t)B_i^{n-i}(t) + tB_{i-1}^{n-1}(t)$*; wir definieren* $B_i^n(t) \equiv 0$ *für* $i < 0$ *oder* $i > n$.

(vii) ***Ableitung:***

$$\frac{d}{dt}B_i^n(t) = n\left(B_{i-1}^{n-1}(t) - B_i^{n-1}(t)\right)$$

mit $B_{-1}^{n-1}(t) \equiv B_n^{n-1}(t) \equiv 0$.

(viii) ***Basis:*** *Die* Bernstein*-Polynome* $\{B_0^n,\ldots,B_n^n\}$ *bilden eine Basis von* $\mathbb{P}_n$.

Beweis. Vergleiche [Deuflhard] S. 228f. Satz 7.31. □

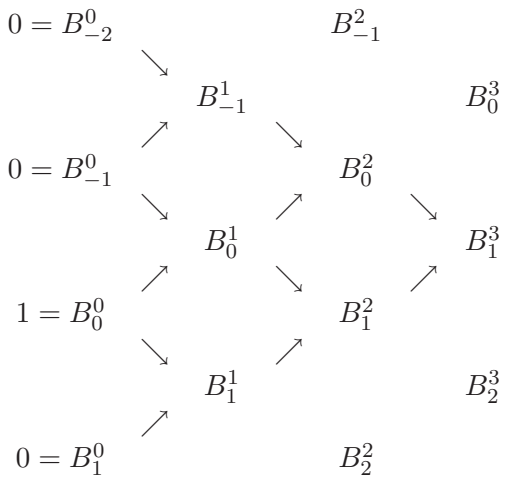
Die Gleichung (4.24) liefert $B_0^0(t) = 1$. Aus der Eigenschaft 4.2.3.(vi) gewinnen wir die linearen und quadratischen *Bernstein*-Polynome

$$\begin{aligned}
B_0^1(t) &= (1-t)B_0^0(t) + tB_{-1}^0(t) = 1-t,\\
B_1^1(t) &= (1-t)B_1^0(t) + tB_0^0(t) = t,\\
B_0^2(t) &= (1-t)B_0^1(t) + tB_{-1}^1(t) = (1-t)^2,\\
B_1^2(t) &= (1-t)B_1^1(t) + tB_0^1(t) = 2t(1-t),\\
B_2^2(t) &= (1-t)B_2^1(t) + tB_1^1(t) = t^2.
\end{aligned} \tag{4.25}$$

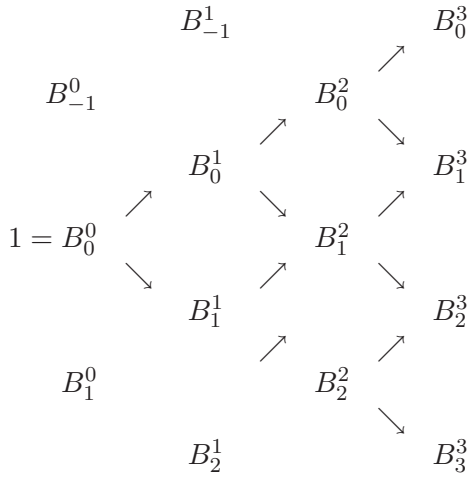
Die Eigenschaft 4.2.3.(vi) liefert einen einfachen Algorihmus, um Werte der *Bernstein*-Polynome zu einem gegebenen t zu bestimmen. In Tabelle 4.1 ist dargestellt, welche B_k^0 benötigt werden, um B_1^3 zu berechnen.
Berücksichtigt man die Nulleinträge ($B_{-2}^0 = B_{-1}^0 = B_1^0 = 0$), d.h. man vernachlässigt die Terme von denen man weiß, das sie verschwinden, so lassen sich alle kubischen *Bernstein*-Polynome, wie in der folgenden Tabelle 4.2 dargestellt, effizient bestimmen.
Die Funktion `AllBernstein.m` kombiniert das in Tabelle 4.2 dargestellte Vorgehen mit Gleichung (4.24) um die Bernstein-Polynome n-ten Grades an einer gegeben Stelle t zu bestimmen.

MATLAB-Funktion: AllBernstein.m

```
1 function B = AllBernstein(n,x)
2 % Compute all n-th Bernstein polynomials
```



Tab. 4.1: Berechnung von B_1^3.



Tab. 4.2: Berechnung aller kubischen Bernstein-Polynome.

```
3  B = zeros(n+1,1);
4  B(1) = 1;
5  for j=1:n
6    saved = 0;
7    for k=1:j
8      temp = B(k);
9      B(k) = saved + (1-x) * temp;
10     saved = x * temp;
11   end
12   B(j+1) = saved;
13 end
```

MATLAB-Beispiel:

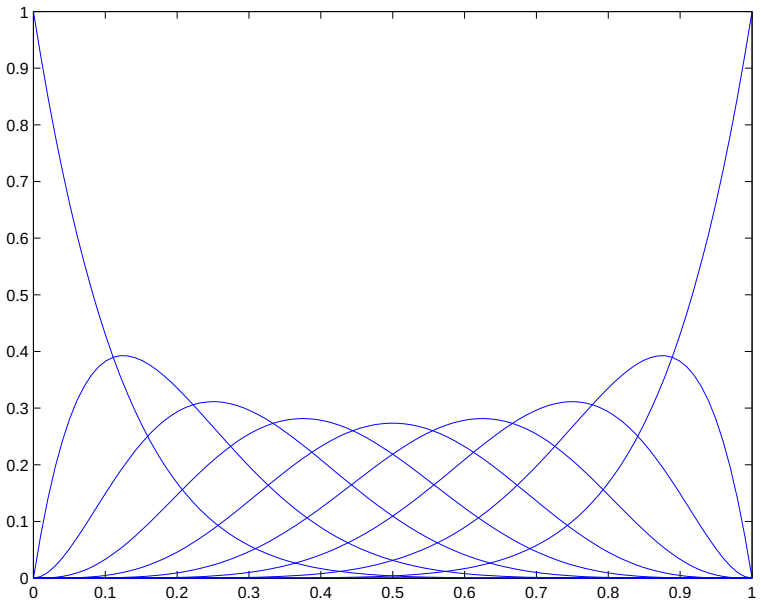


Abb. 4.5: Ergebnis der Matlab-Zeilen.

Die folgenden Zeilen stellen die Bernstein-Polynome $B_0^8, \ldots, B_8^8$ graphisch dar.

```
>> n = 8;  no = 100;
>> t = linspace(0,1,no);
>> B = zeros(n+1,no);
>> for k=1:no,B(:,k)=AllBernstein(n,t(k));
   end
>> hold on, for k=1:n+1, plot(t,B(k,:)),
   end
```

Bemerkung 4.2.4 (*Bézier*-Darstellung eines Polynoms) *Nach Satz 4.2.3 bilden die* Bernstein-*Polynome* $\{B_0^n, \ldots, B_n^n\}$ *eine Basis des* $\mathbb{P}_n$*. Daher lässt sich jedes Polynom vom Grad kleiner gleich* n *als Linearkombination dieser* Bernstein-*Basis-Polynome darstellen*

$$P_n(t) = \sum_{j=0}^{n} \beta_j B_j^n(t).$$

Die β_i *heißen* **Bézier-*Koeffizienten*** *und diese Linearkombination wird* **Bézier-*Darstellung*** *des Polynoms* P_n *genannt.*

4.2.3 Bézierkurven

Kommen wir nun zur parametrisierten Darstellung einer Kurve im $\mathbb{R}^d$.

$$\mathbf{C}(t) = (x_1(t), \ldots, x_d(t))^T.$$

Im d-dimensionalen kann man die Komponenten einer Kurve jeweils (analog zum eindimensionalen) auch als Linearkombination der *Bernstein*-Basis-Polynome darstellen. Diese Darstellung ist dann die so genannten **Bézierkurve**:

Definition 4.2.5 (Bézierkurve) *Gegeben seien* $n+1$ **Kontrollpunkte** $\mathbf{P}_j = (x_1, \ldots, x_d)^T \in \mathbb{R}^d$*,* $d \in \mathbb{N}$*. Eine* **Bézierkurve** n*-ten Grades zu gegebenen* $n+1$ *Punkten* $\mathbf{P}_j \in \mathbb{R}^d$*,* $j = 0, \ldots, n$*,*

$d \in \mathbb{N}$, ist für $t \in [0,1]$ definiert als

$$\mathbf{C}(t) = \sum_{j=0}^{n} B_j^n(t)\,\mathbf{P}_j\,. \tag{4.26}$$

Bemerkung 4.2.6 *Da die Bernstein-Polynome eine Zerlegung der Eins bilden, ist die Summe ausgewertet für ein festes t nichts anderes als die Linearkombination der gegebenen Punkte $\mathbf{P}_j$.*

Bemerkung 4.2.7 (Kontrollpolygon) *Ist $d = 2$, so heißt die geradlinige Verbindung der Punkte $\mathbf{P}_0, \ldots, \mathbf{P}_n$ das **Kontrollpolygon**.*

4.2.4 Der *de Casteljau*-Algorithmus

Wir wollen nun den ***de Casteljau*-Algorithmus** einführen, welcher auf fortgesetzten Konvexkombinationen beruht. Dieser Algorithmus liefert die Grundlage für die Bedeutung der Bézierkurven. Einerseits liefert er den Funktionswert $\mathbf{C}(t)$ einer Kurve an einer beliebigen Stelle und die Ableitung. Andererseits ermöglicht er die schnelle Berechnung der Bézierkurve durch Unterteilung in Teilsegmente.

Herleitung des Algorithmus: Betrachtet man eine Bézierkurve auf dem Intervall $[0,1]$, so ist es möglich, diese in zwei neue Bézierkurven zu unterteilen: eine auf dem Teilintervall $[0,t]$ und die andere auf dem Teilintervall $[t,1]$.

Für $n = 2$ und $\mathbf{C}(t) = \sum_{j=0}^{2} B_j^2(t)\mathbf{P}_j$ gilt

$$\begin{aligned}\mathbf{C}(t) &= (1-t)^2\mathbf{P}_0 + 2t(1-t)\mathbf{P}_1 + t^2\mathbf{P}_2 \\ &= (1-t)\Big(\underbrace{(1-t)\mathbf{P}_0 + t\,\mathbf{P}_1}_{\text{linear}}\Big) + t\Big(\underbrace{(1-t)\mathbf{P}_1 + t\,\mathbf{P}_2}_{\text{linear}}\Big)\,.\end{aligned}$$

Somit kann $\mathbf{C}(t)$ als Linearkombination von zwei Bézierkurven 1-ten Grades bestimmt werden. Betrachten wir dies nun allgemeiner. Bezeichnen wir eine beliege Bézierkurve n-ten Grades mit

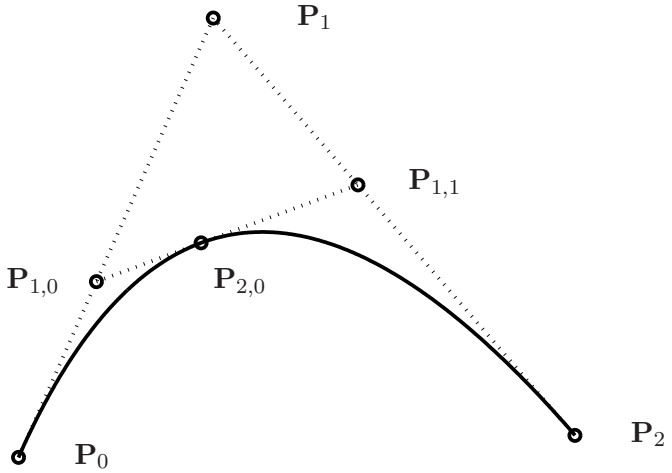


Abb. 4.6: Berechnung eines Punktes durch wiederholte lineare Interpolation für $t = 2/5$.

$\mathbf{C}_n(t : \mathbf{P}_0, \ldots, \mathbf{P}_n)$, dann liefert uns die Rekursion 4.2.3.(vi) die Darstellung

$$\mathbf{C}_n(t : \mathbf{P}_0, \ldots, \mathbf{P}_n) = (1-t)\mathbf{C}_{n-1}(t : \mathbf{P}_0, \ldots, \mathbf{P}_{n-1}) + t\mathbf{C}_{n-1}(t : \mathbf{P}_1, \ldots, \mathbf{P}_n)\,.$$

Dies liefert ein rekursives Verfahren zur Bestimmung von $\mathbf{C}(t_0) = \mathbf{P}_{n,0}(t_0)$ auf einer Bézierkurve n-ten Grades, nämlich

$$\mathbf{P}_{k,j}(t_0) = (1-t_0)\mathbf{P}_{k-1,j}(t_0) + t_0\mathbf{P}_{k-1,j+1}(t_0) \quad \text{für} \begin{cases} k &= 1,\ldots,n \\ i &= 0,\ldots,n-k \end{cases} \tag{4.27}$$

Die Gleichung wird als ***de Casteljau*-Algorithmus** bezeichnet.

MATLAB-Funktion: deCasteljau1.m

```
function C = deCasteljau1(P,t)
% Compute point on a bezier curve using Casteljau
for k=1:size(P,2)-1
  for i=1:size(P,2)-k
    P(:,i) = (1-t)*P(:,i) + t*P(:,i+1);
  end
end
C = P(:,1);
```

MATLAB-Beispiel:

Die folgenden Zeilen liefern das Kontrollpolygon und die Bézierkurve zu den Punkten $P_0 = (0,0)$, $P_0 = (1,-1)$, $P_0 = (3,4)$ und $P_3 = (3,3)$.

```
>> P=[0, 1, 2, 3;
>>    0 -1, 4, 3];
>> t = linspace(0,1,100);
>> for k=1:length(t)
>>   C(:,k)= deCasteljau1(P,t(k));
>> end
>> plot(C(1,:),C(2,:),'k-',
        P(1,:),P(2,:),'r*:');
```

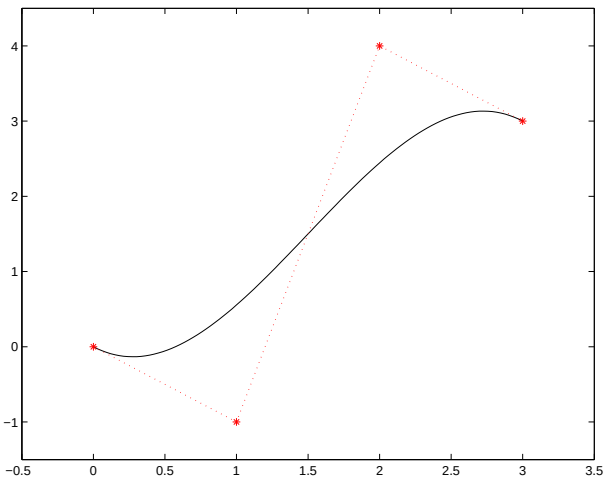


Abb. 4.7: Kontrollpolygon und Bézierkurve, Ergebnis des Matlab-Beispiels.

Bemerkung 4.2.8 (Matrix-Matrix-Vektor Darstellung des *de Casteljau*-Algorithmus) *Der* de Casteljau-*Algorithmus kann algorithmisch günstig als Matrix-Matrix-Vektor-Produkt dargestellt werden. Sei $\mathbf{C}(t)$ eine Bézierkurve n-ten Grades, dann gilt*

$$\mathbf{C}(t) = \begin{pmatrix} \mathbf{P}_0 & \mathbf{P}_1 & \dots & \mathbf{P}_n \end{pmatrix} \boldsymbol{B}_n \begin{pmatrix} t^n \\ t^{n-1} \\ \vdots \\ t \\ 1 \end{pmatrix}$$

mit einer für jedes n konstanten Matrix $\boldsymbol{B}_n$ mit Komponenten

$$b_{i+1,k+1} = \begin{cases} (-1)^{i+k+n} \begin{pmatrix} n-i \\ k \end{pmatrix} \begin{pmatrix} n \\ i \end{pmatrix}, & \text{für } i = 0,1,\dots,n, \quad k = 0,1,\dots,n-1, \\ 0, & \text{sonst.} \end{cases}$$

Kommen wir nochmals auf den Vergleich der Darstellungen zurück, d.h. die Koordinatenfunktionen $x(t)$, $y(t)$ und $z(t)$ sind Polynome in monomialer Basis oder in der Bernstein-Basis. Betrachten wir diesbezüglich einige Beispiele.

Beispiel 4.2.9 Es sei $n = 1$. Aus (4.25) erhalten wir $B_0^1(t) = 1 - t$ und $B_1^1(t) = t$. Die Darstellung (4.26) nimmt dann die Form

$$\mathbf{C}(t) = (1-t)\mathbf{P}_0 + t\mathbf{P}_1$$

an. Dies ist eine gerade Linie von $\mathbf{P}_0$ nach $\mathbf{P}_1$.

Beispiel 4.2.10 Es sei $n = 2$. Aus (4.25) und (4.26) erhalten wir

$$\mathbf{C}(t) = (1-t)^2\mathbf{P}_0 + 2t(1-t)\mathbf{P}_1 + t^2\mathbf{P}_2.$$

Dies ist eine parabolische Kurve von $\mathbf{P}_0$ nach $\mathbf{P}_2$ (siehe Abb. 4.8 rechts). Man beachte, dass der Polygonzug mit den Punkten $\mathbf{P}_0, \mathbf{P}_1, \mathbf{P}_2$, sprich das Kontrollpolygon, die Form der Kurve näherungsweise gut approximiert. Für die Endpunkte gilt $\mathbf{P}_0 = \mathbf{C}(0)$ und $\mathbf{P}_2 = \mathbf{C}(1)$. Die tangentialen Richtungen an den Endpunkten sind parallel zu $\mathbf{P}_1 - \mathbf{P}_0$ und $\mathbf{P}_2 - \mathbf{P}_1$. Die Kurve liegt im Dreieck mit den Ecken $\mathbf{P}_0, \mathbf{P}_1, \mathbf{P}_2$.

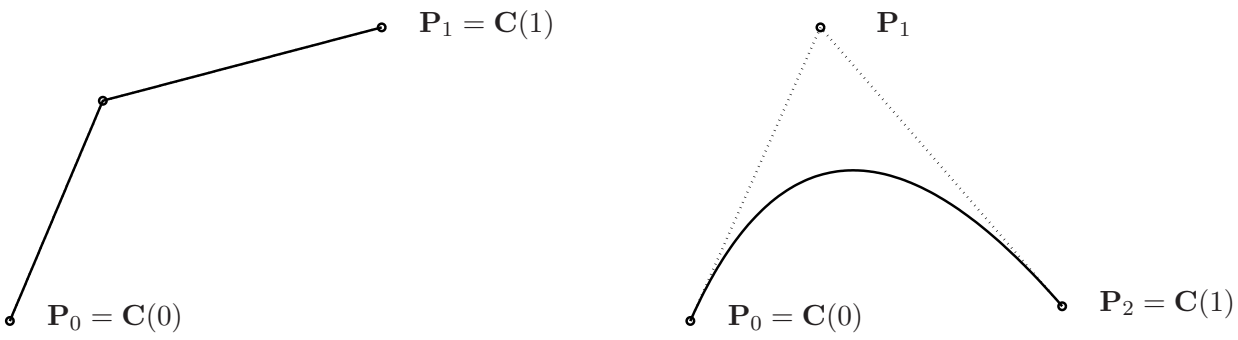


Abb. 4.8: Eine lineare (links) und quadratische (rechts) Bézierkurve.

Beispiel 4.2.11 Es sei $n = 3$. Wir erhalten

$$\mathbf{C}(t) = (1-t)^3\mathbf{P}_0 + 3t(1-t)^2\mathbf{P}_1 + 3t^2(1-t)\mathbf{P}_2 + t^3\mathbf{P}_3.$$

Beispiele kubischer Bézierkurven sind in Abb. 4.9 dargestellt. Man beachte, dass das Kontrollpolygon näherungsweise die Kurve beschreibt. Für die Endpunkte gilt $\mathbf{P}_0 = \mathbf{C}(0)$ und $\mathbf{P}_3 = \mathbf{C}(1)$. Die tangentialen Richtungen an den Endpunkten sind parallel zu $\mathbf{P}_1 - \mathbf{P}_0$ und $\mathbf{P}_3 - \mathbf{P}_2$. Die Kurve liegt in der konvexen Hülle der Punkte $\mathbf{P}_0, \mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3$. Keine Gerade schneidet die Kurve häufiger als sie das Kontrollpolygon schneidet. Die Kurve krümmt sich bei $t = 0$ in die gleiche Richtung wie $\mathbf{P}_0, \mathbf{P}_1, \mathbf{P}_2$ bzw. bei $t = 1$ wie $\mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3$.

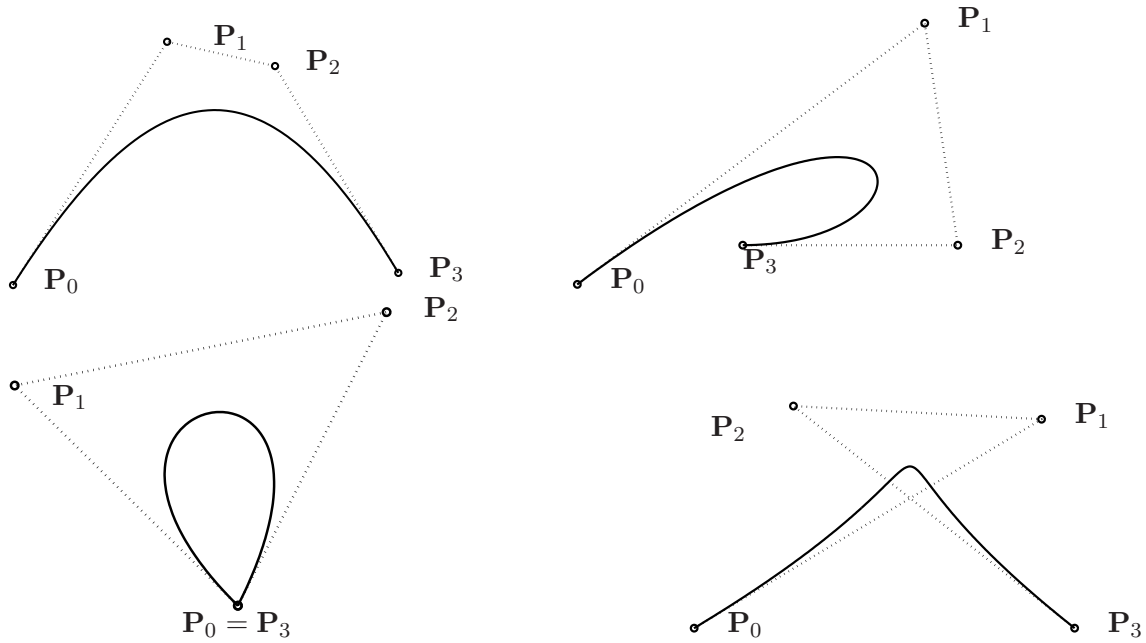


Abb. 4.9: Kubische Bézierkurve und zugehörige Kontrollpolygone.

4.2.5 Bézierflächen

Die Kurve $\mathbf{C}(t)$ ist eine vektorwertige Funktion in einer Variablen. Eine Fläche ist eine vektorwertige Funktion in zwei Parametern s und t und stellt die Abbildung eines Gebiets R der st-Ebene in den 3-dimensionalen Raum dar, nämlich

$$\mathbf{S}(s,t) = (x(s,t), y(s,t), z(s,t)))^T, \quad (s,t) \in R.$$

Es gibt mehrere Möglichkeiten, die Koordinatenfunktionen zu definieren. Der sicherlich einfachste und häufig verwendete Ansatz, ist der des Tensorprodukts, d.h.

$$\mathbf{S}(s,t) = (x(s,t), y(s,t), z(s,t))) = \sum_{i=0}^{m} \sum_{j=0}^{n} f_i(s) g_j(t) \mathbf{b}_{ij}$$

mit

$$\mathbf{b}_{ij} = (x_{ij}, y_{ij}, z_{ij}) \quad 0 \le s,t \le 1\,.$$

Man beachte, dass die Definitionsmenge dieser Abbildung das Quadrat $[0,1]^2$ ist. Verwendet man als Basisfunktionen wieder die Bernstein-Polynome so erhält man

$$\mathbf{S}(s,t) = \sum_{i=0}^{m} \sum_{j=0}^{n} B_i^m(s) B_j^n(t) \mathbf{P}_{ij} \quad 0 \le s,t \le 1\,. \tag{4.28}$$

Für ein festes s_0 gilt

$$\begin{aligned}\mathbf{C}_{s_0}(t) = \mathbf{S}(s_0,t) &= \sum_{i=0}^{m}\sum_{j=0}^{n} B_i^m(s_0)B_j^n(t)\mathbf{P}_{ij} \\ &= \sum_{j=0}^{n} B_j^n(t)\left(\sum_{i=0}^{m} B_i^m(s_0)\mathbf{P}_{ij}\right) \\ &= \sum_{j=0}^{n} B_j^n(t)\mathbf{Q}_j(s_0), \qquad (4.29)\end{aligned}$$

wobei $\mathbf{Q}_j(s_0) = \sum_{i=0}^{m} B_i^m(s_0)\mathbf{P}_{ij}$, $j = 0,\ldots,n$ eine Bézierkurve ist, die auf der Fläche liegt.

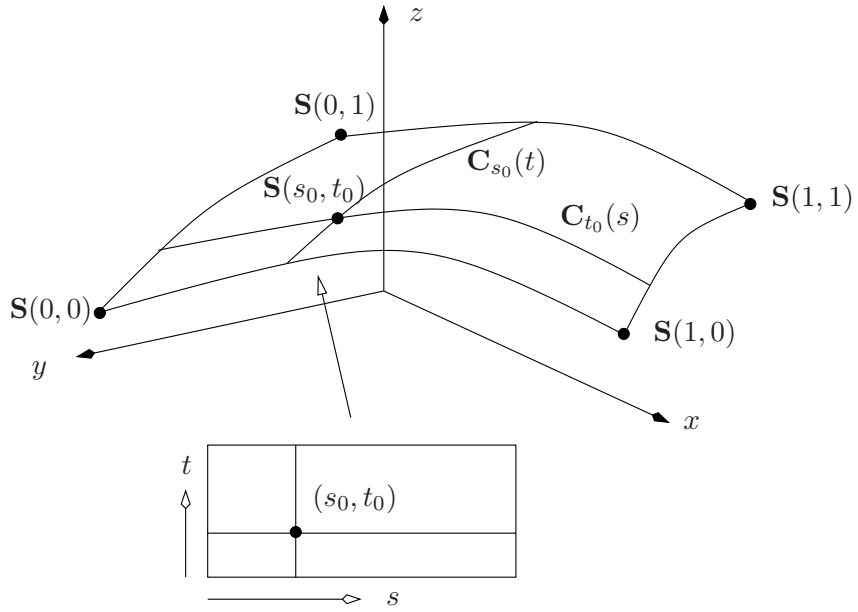
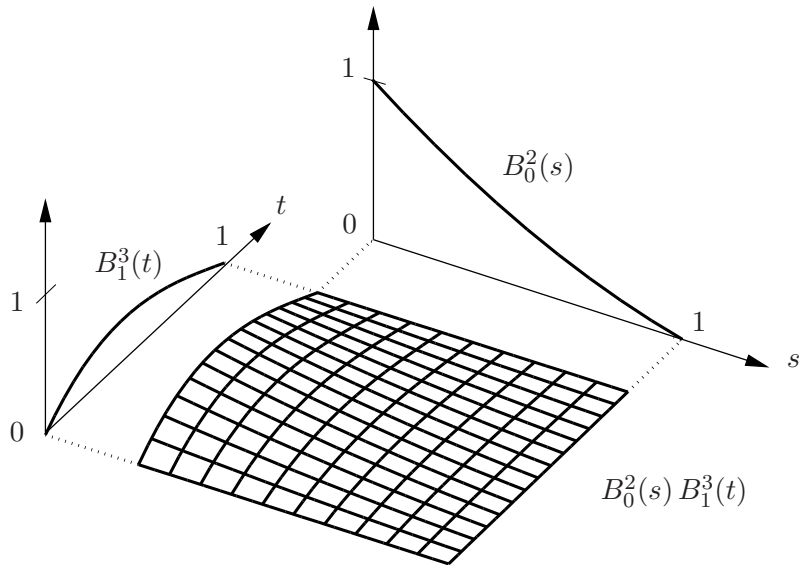


Abb. 4.10: Eine Tensorproduktfläche und isoparametrische Kurven.



Mittels (4.29) kann man also (4.28) zu gegebenen (s_0,t_0) durch mehrmaliges Anwenden des eindimensionalen *de Casteljau*-Algorithmus bestimmen. Dieses Vorgehen ist in der Routine `deCasteljau2.m` realisiert.

MATLAB-Funktion: deCasteljau2.m

```
function S = deCasteljau2(P,s,t)
% Compute a point on a Bezier surface
n = size(P,2); m = size(P,3);
if n <= m
  for j = 1:m
    Q(:,j) = deCasteljau1(squeeze(P(:,:,j)),s);
  end
  S=deCasteljau1(Q,t);
else
  for i = 1:n
    Q(:,i)=deCasteljau1(squeeze(P(:,i,:)),t);
  end
  S=deCasteljau1(Q,s);
end
```

In Abb. 4.11 ist das Kontrollnetz und die Bézierfläche beispielhaft dargestellt.

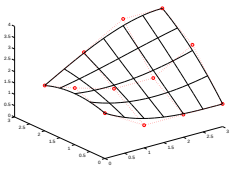


Abb. 4.11: Beispiel eines Kontrollnetz und Bézierfläche in $\mathbb{R}^2$.

4.3 GRUNDLEGENDE ALGORITHMEN

MATLAB-Funktion: CurveKnotIns.m

```
function [U,Q] = CurveKnotIns(p,U,P,t,k,s,r)
if p < s+r, Q=P; return, end
% compute new curve from knot insertion
np = length(U)-p-1;
% unaltered control points
Q = P(:,1:k-p);
Q(:,k-s+r:np+r)=P(:,k-s:np);
R = P(:,k-p:k-s);
for j=1:r % insert new knot r times
   L=k-p+j-1;
   for i=1:p-j-s+1
     alpha = (t-U(L+i))/(U(i+k)-U(L+i));
     R(:,i) = alpha*R(:,i+1)+(1-alpha)*R(:,i);
   end
   Q(:,L+1) = R(:,1);
```

```
    Q(:,k+r-j-s)= R(:,p-j-s+1);
end
%copy remaining control points
Q(:,L+2:k-s)= R(:,2:k-s-L);
% new knot vevtor
U = [U(1:k),t*ones(1,r),U(k+1:end)];
```

MATLAB-Funktion: CurveSplit.m

```
function [U1,P1,U2,P2] = CurveSplit(p,U,P,t)
if t==U(1)
  U1=[];P1=[];U2=U;P2=P;return
elseif t==U(end)
  U1=U;P1=P;U2=[];P2=[];return
end
k = FindSpan(p,U,t);
s = sum((t==U));
if p>s
  [U,P]=CurveKnotIns(p,U,P,t,k,s,p-s);
end
U1 = U([1:k+p-s,k+p-s])-U(1);
U1 = U1/max(U1); P1 = P(:,1:k);
U2 = U([k+1-s,k+1-s:end])-U(k+1-s);
U2 = U2/max(U2); P2 = P(:,k-s:end);
```

MATLAB-Beispiel:

Die folgenden Zeilen stellen die Bernstein-Polynome $B_0^8, \ldots, B_8^8$ graphisch dar.

```
>> P =[0,1,5,5;
       0,2,0,1];
>> U = [0,0,0,0,1,1,1,1];
>> p = 3;
>> t = 1/3;
>> [U1,P1, U2, P2] = CurveSplit(p,U,P,t)
U1 =
   0  0  0  0  1  1  1  1
P1 =
         0  0.3333  1.0000  1.7407
         0  0.6667  0.8889  0.9259
U2 =
   0  0  0  0  1  1  1  1
P2 =
   1.7407  3.2222  5.0000  5.0000
   0.9259  1.0000  0.3333  1.0000
```

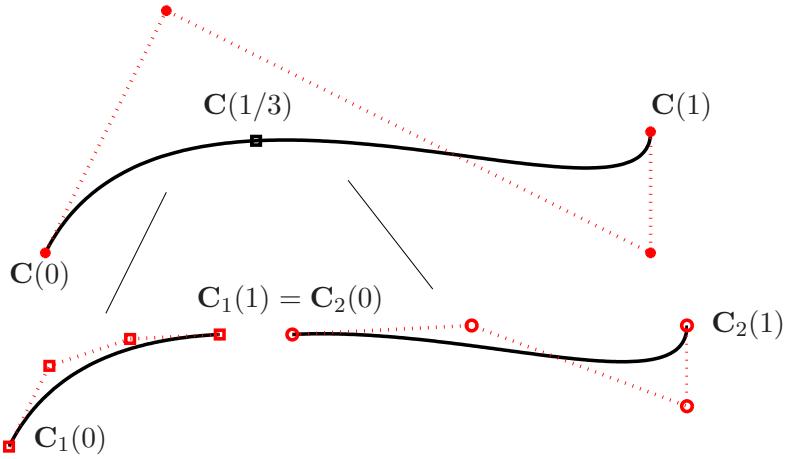


Abb. 4.12: Ergebnis der Matlab-Zeilen.

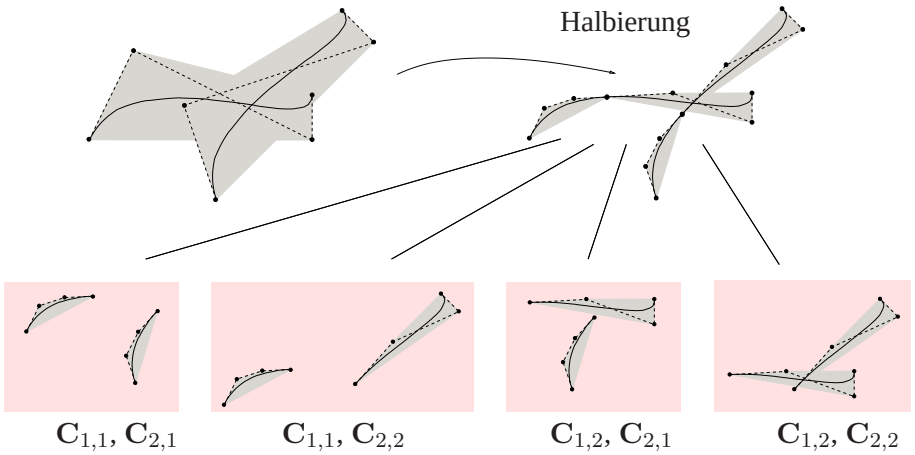


Abb. 4.13: Testen auf gemeinsamen Schnitt der einzelnen konvexen Hüllen nach Halbierung der einzelnen Bézierkurven $\mathbf{C}_1 = \mathbf{C}_{1,1} \cup \mathbf{C}_{1,2}$ und $\mathbf{C}_2 = \mathbf{C}_{2,1} \cup \mathbf{C}_{2,2}$.

MATLAB-Funktion: isLine.m

```
function [flag,P0,P1] = isLine(P,tol1,tol2)
S = mean(P,2);
PmS = P-S*ones(1,size(P,2));
[j,k] = max(sum(abs(PmS)));
rot = GivensRotMat(PmS(1,k),PmS(2,k));
Q = rot'*([PmS(1,:);PmS(2,:)]);
h = max(Q,[],2)-min(Q,[],2);
if h(2) <= max(tol1,tol2*h(1))
  flag = 1;
  P0 = S + rot*[h(1);0]/2;
  P1 = S + rot*[-h(1);0]/2;
else
  flag = 0; P0=[];P1=[];
end
```

MATLAB-Funktion: LineIntersect.m

```
function [flag,s] = LineIntersect(x,y)
% check for intersection of two line segments
% 0 no intersection, 1 one or more intersection points
dx = x(:,2)-x(:,1); mx = norm(dx);
dy = y(:,2)-y(:,1); my = norm(dy);
dw = (y(:,2)+y(:,1)-x(:,2)-x(:,1)); mw = norm(dw);
flag = 1;s=[];
if abs(det([dx,dy])) >= 1e-12 *mx*my % check for non parallel
  if ~sum(abs([dx,dy]\dw)>1)
    t = [dx,dy]\dw;
    s = ((x(:,2)+x(:,1))+t(1)*dx)/2;
    return
  end
elseif det([dx,dw]) < max(1e+10*realmin,1e-12*mx*mw)  % on common
    line
  mm = [((x(:,2)+x(:,1))/2)*[1,1],((y(:,2)+y(:,1))/2)*[1,1]];
  dd = [dx,dx,dy,dy];
  xx = [dw+dy,dw-dy,-dw+dx,-dw+dx];
  for j=1:4
    t = xx(:,j)'*dd(:,j)/(dd(:,j)'*dd(:,j));
    if abs(t)<=1
      s = mm(:,j)+t/2*dd(:,j);
      return
    end
  end
end
flag = 0;
```

MATLAB-Funktion: comConvHull.m

```
function flag = comConvHull(xy,st)
if comBoundingBoxes(min(xy,[],2),max(xy,[],2), ...
                    min(st,[],2),max(st,[],2))
  flag = 1;

  if inConvHull(xy,mean(st(:,1:end-1),2)), return, end
  if inConvHull(st,mean(xy(:,1:end-1),2)), return, end

  for j = 1:size(xy,2)-1
    for k = 1:size(st,2)-1
      if LineIntersect(xy(:,j:j+1),st(:,k:k+1))
        return
      end
    end
  end
end
flag = 0;
```

MATLAB-Funktion: CurveBisection.m

```
function points = CurveBisection(p1,U1,Q1,p2,U2,Q2,tol)
[flag1,a1,b1] = isLine(Q1,tol(1),tol(2));
[flag2,a2,b2] = isLine(Q2,tol(1),tol(2));
if flag1 && flag2  % both segments are lines
  [flag,points] = LineIntersect([a1,b1],[a2,b2]);
else
  xy = myConvHull(Q1,flag1,a1,b1);
  st = myConvHull(Q2,flag2,a2,b2);
  points =[];
  if  comConvHull(xy,st) % intersection of convex hulls non empty
    [U11,Q11,U21,Q21] = CurveSplit(p1,U1,Q1,(max(U1)-min(U1))/2);
    [U12,Q12,U22,Q22] = CurveSplit(p2,U2,Q2,(max(U2)-min(U2))/2);
    points = [points,CurveBisection(p1,U11,Q11,p2,U12,Q12,tol)];
    points = [points,CurveBisection(p1,U11,Q11,p2,U22,Q22,tol)];
    points = [points,CurveBisection(p1,U21,Q21,p2,U12,Q12,tol)];
    points = [points,CurveBisection(p1,U21,Q21,p2,U22,Q22,tol)];
  end
end

function h = myConvHull(Q,flag,a,b)
% Q
% [flag,a,b] = isLine(Q,1e-7,1e-7)

if flag
  h = [a,b,a];
else
  h = Q(:,convhull(Q(1,:),Q(2,:)));
end
```

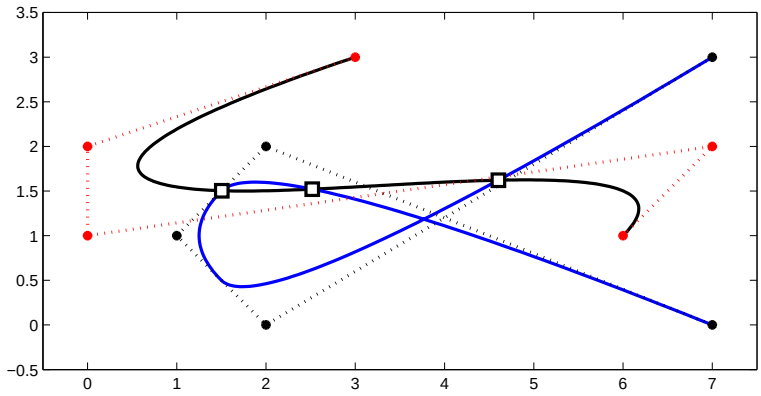


Abb. 4.14: Ergebnis der Matlab-Zeilen.

Die folgenden Zeilen stellen die Bernstein-Polynome $B_0^8, \ldots, B_8^8$ graphisch dar.

MATLAB-Beispiel:

```
>> P1=[3 0,0,7,6;
       3,2,1,2,1]; p1=3;
>> U1 = [0,0,0,0,1/2,1,1,1,1];
>> P2 = [7,2,1,2,7;
        3,0,1,2,0]; p2=2;
>> U2 = [0,0,0,1/3,2/3,1,1,1];
>> CurveBisection(p1,U1,P1,p2,U2,P2, ...
                  [1e-9,1e-7])
ans =
    1.5038    4.6028    2.5178
    1.5037    1.6214    1.5214
```

4.4 B-SPLINES

In Abschnitt 4.1 haben wir uns mit den kubischen Splines beschäftigt und diese mit einem direkten Ansatz der Form

$$s_i(x) = a_i(x - x_i)^3 + b_i(x - x_i)^2 + c_i(x - x_i) + d_i,$$

hergeleitet. Bis auf eine Verschiebung um x_i ist dies ein monomialer Ansatz. Im Folgenden wollen wir uns mit einem allgemeineren[4] Ansatz beschäftigen, der durch die Anwendung in der Computergrafik motiviert ist.

Wir hatten bereits gesehen, dass die Dimension des Splineraums $dim\, \mathcal{S}^k(\mathcal{T}) = k + n =: \ell$ ist. Die Frage ist nun: Kann man eine einfach handhabbare Basis für $\mathcal{S}^k(\mathcal{T})$ konstruieren? Wir suchen nun eine geeignete Basis, die ähnlich gute Eigenschaften wie die Bernstein-Polynome hat.

Zunächst halten wir aber der Vollständigkeit halber fest, dass es eine numerisch nicht brauchbare Basis aus den Monomen und abgebrochenen Potenzen gibt:

[4] Allgemeiner bzgl. des Polynomgrads auf jedem Teilintervall, als auch der Glattheitsanforderung an den Knoten zwischen den Teilintervallen, bzw. an den beiden Endpunkten.

Bemerkung 4.4.1 (Basis für den Splineraum) *Für die* **abgebrochenen Potenzen**

$$x_+^\nu := \begin{cases} x^\nu, & \text{falls } x \geq 0, \\ 0, & \text{falls } x < 0, \end{cases}$$

gilt $(x - x_i)_+^k \in \mathcal{S}^k(\mathcal{T})$, $i = 1 \ldots, n-1$. *Es gilt sogar, dass*

$$\{x^i,\ i = 0, \ldots, k\} \cup \{(x - x_i)_+^k,\ i = 1, \ldots, n-1\} \tag{4.30}$$

eine Basis für $\mathcal{S}^k(\mathcal{T})$ *bildet (*Übung*). Diese ist jedoch numerisch unbrauchbar, denn*

- *sie besteht aus „globalen" Funktionen,*
- *sie ist äußest schlecht konditioniert.*

Wir werden in diesem Abschnitt eine Basis für den Splineraum $\mathcal{S}^k(\mathcal{T})$ einführen, deren Basisfunktionen einen Träger minimaler Länge haben (Monome haben ganz $\mathbb{R}$ als Träger) und deren Elemente sich effektiv und numerisch stabil berechnen lassen. Wir erinnern daran, dass mit dem **Träger** (engl. support) einer Funktion $f : \mathbb{R} \to \mathbb{R}$ die Menge

$$\operatorname{supp}(f) := \overline{\{x \in \mathbb{R}, f(x) \neq 0\}}$$

bezeichnet wird, wobei der Strich den Abschluß der Menge bezeichnet. Splines, die in dieser Basis dargestellt werden, nennt man **B-Splines**. Der heutige Erfolg der Splines hängt i.W. an der Konstruktion dieser Basis.

Vorab betrachten wir allerdings zwei Spezialfälle, die unterschiedliche Basen veranschaulichen:

Beispiel 4.4.2 (Basis von $\mathcal{S}^0(\mathcal{T})$) Gegeben sei eine Knotenfolge $\mathcal{T} = \{x_0 = 0, x_1, x_2, x_3\}$. Für $k = 0$ bedeutet $C^{k-1}([a,b]) = C^{-1}([a,b])$, dass es sich um unstetige Funktionen aus $\mathbb{P}_k = \mathbb{P}_0$ handelt, also um stückweise Konstanten. Für $k = 0$, $n = 3$ und $x_0 = 0$ kann man leicht zwei Basen von $\mathcal{S}^0(\mathcal{T})$ angeben:

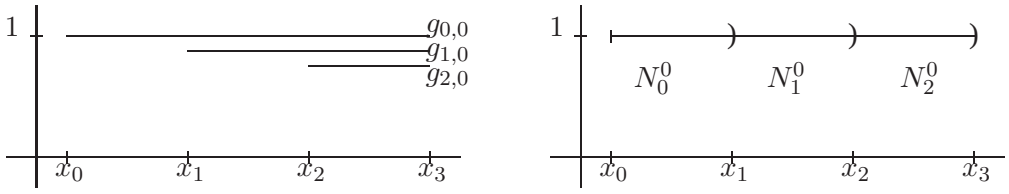


Abb. 4.15: Zwei Basen von $\mathcal{S}^0(\mathcal{T})$.

$$\begin{aligned} g_{0,0}(x) &= x^0 \equiv 1 = (x - x_0)_+^0, \\ g_{1,0}(x) &= (x - x_1)_+^0, \\ g_{2,0}(x) &= (x - x_2)_+^0, \end{aligned} \qquad N_i^0 = \chi_{[x_i, x_{i+1})}, \quad i = 0, 1, 2.$$

wobei

$$\chi_{[x_i, x_{i+1})} := \begin{cases} 1, & \text{falls } x_i \leq x < x_{i+1} \\ 0 & \text{sonst.} \end{cases}$$

die Indikatorfunktion auf dem halb offenen Intervall $[x_i, x_{i+1})$ bezeichnet. Offenbar ist die zweite Basis „lokaler": Die Träger von $\{N_{i,1} : i = 0, 1, 2\}$ sind bezüglich der Knoten x_i minimal.

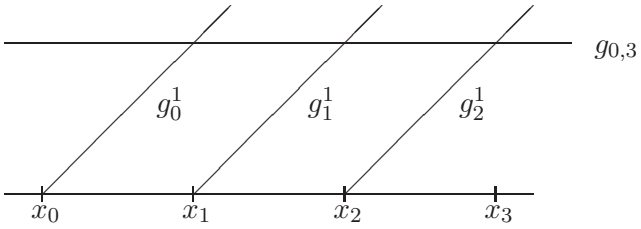


Abb. 4.16: Erste Basis für $\mathcal{S}^1(\mathcal{T})$ gemäß (4.30).

Beispiel 4.4.3 (Basis von $\mathcal{S}^1(\mathcal{T})$) Gegeben sei eine Knotenfolge $\mathcal{T} = \{x_0, x_1, x_2, x_3\}$. Für $k = 1$ besteht $\mathcal{S}^2(\mathcal{T})$ aus stückweise linearen stetigen Funktionen. Für $n = 3$ erhalten wir wiederum zwei Basen, zunächst gemäß (4.30)

$$\{x^0, x^1\} \cup \{(x - x_1)_+, (x - x_2)_+\}.$$

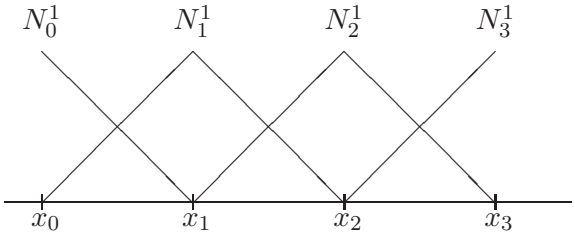


Abb. 4.17: Zweite Basis für $\mathcal{S}^1(\mathcal{T})$ bestehend aus „Hutfunktionen“.

Mit der folgenden Umbenennung $t_i := x_i, i = 0, \ldots, n$ und zwei Hilfsknoten $t_{-1} < t_0$, und $t_4 > t_3$ (deren Lage auf $[a, b]$ keinerlei Einfluss hat), definiert man

$$N_j^1(x) := \begin{cases} \frac{x - t_{j-1}}{t_j - t_{j-1}}, & t_{j-1} \le x \le t_j, \\ \frac{t_{j+1} - x}{t_{j+1} - t_j}, & t_j \le x \le t_{j+1}, \\ 0, & \text{sonst.} \end{cases} \tag{4.31}$$

Man kann sich leicht überlegen, dass die so definierten $\{N_j^1 \,;\, j = 0, \ldots, n\}$ eine Basis von $\mathcal{S}^1(\mathcal{T})$ bilden: Zunächst gilt $N_j^2 \in \mathcal{S}^1(\mathcal{T})$ und die Kardinalität ist $n + 1 = \dim \mathcal{S}^1(\mathcal{T})$. Bleibt noch die lineare Unabhängigkeit zu zeigen. Angenommen, es gelte

$$\sum_{j=0}^{n} c_j N_j^1(x) = 0$$

für alle $x \in [a,b]$. Dann gilt dies insbesondere für alle $x = t_i$, $i = 0,\ldots,n$. Wegen

$$N_j^1(t_i) = \delta_{ij}$$

folgt $c_j = 0$ für alle $j = 0,\ldots,n$.

4.4.1 Rekursive Definition der B-Splines-Basisfunktionen

Wie bereits erwähnt, will man analog zu Bernstein-Polynomen eine Basis für den Splineraum konstruieren, die ähnlich gute Eigenschaften hat. D.h. insbesondere einen lokalen Träger. Außerdem soll die Darstellung bzgl. dieser Basis die Nachteile der Bézier-Kurven überwinden. Bézier-Kurven haben zwei wesentliche Nachteile:

- Der Aufwand zur Berechnung ist groß, wenn man viele Kontrollpunkte hat.
- Die Kontrollpunkte haben globalen Einfluss auf den Kurvenverlauf.

B-Splines sind hierbei, wie wir sehen werden, eine Verallgemeinerung von Bèzierkurven, bei denen die Kontrollpunkte nur noch lokalen Einfluss haben. Wir definieren so genannte *B-Spline-Basisfunktionen* rekursiv und zeigen später, dass diese eine Basis des Splineraums bilden. Wir definieren B-Splines durch die folgende Rekursionsformel von *de Boor, Cox* und *Mansfield*:

Definition 4.4.4 (B-Spline-Basisfunktionen) *Sei $\tilde{\mathcal{T}} = \{t_1, t_2, \ldots, t_m\}$ eine nichtfallende Knotenfolge reeller Zahlen, d.h. $t_i \le t_{i+1}$, $i = 1,\ldots,m-1$. Die t_i werden als* **Knoten** *und $\tilde{\mathcal{T}}$ als* **Knotenvektor** *bezeichnet. Die i-***te B-Spline-Basisfunktion vom Grade** k *(Ordnung $k+1$) $N_i^k, k = 0,\ldots,m-1, i = 0,\ldots,m-k-1$ ist definiert für $k = 0$ als stückweise konstante Funktion der Form*

$$N_i^0(x) := \begin{cases} 1 & , \quad \text{falls } t_i \le x < t_{i+1} \\ 0 & , \quad \text{sonst} \end{cases} \tag{4.32}$$

und für $k > 0$ durch

$$N_i^k(x) := \frac{x - t_i}{t_{i+k} - t_i} N_i^{k-1}(x) + \frac{t_{i+k+1} - x}{t_{i+k+1} - t_{i+1}} N_{i+1}^{k-1}(x)\,. \tag{4.33}$$

Bemerkung 4.4.5 (Eigenschaften von B-Spline-Basisfunktionen) *(i) N_i^0 ist eine Treppenfunktion, die auf dem halboffenen Intervall $[t_i, t_{i+1})$ Eins ist und sonst verschwindet. Vergleiche hierzu Abbildung 4.15 rechts.*

(ii) Man beachte die rechtsseitige Stetigkeit in der Definition der N_i^0, d.h.

$$\lim_{x \to t_{i+1}^+} N_i^0(x) = N_{i+1}^0(t_{i+1}).$$

(iii) Für $k > 0$ ist $N_i^k(t)$ eine Linearkombination von zwei Basisfunktionen vom Grade $(k-1)$.

(iv) Die Berechnung einer Menge von Basisfunktionen erfordert einen Knotenvektor $\tilde{\mathcal{T}}$ und einen Grad k.

(v) Das i-te Knotenintervall $[t_i, t_{i+1})$ kann die Länge Null haben, da aufeinanderfolgende Knoten nicht verschieden sein müssen. In (4.33) *kann der Nenner im Bruch Null werden; dieser Quotient sei per Definition Null.*

(vi) Die $N_i^k(t)$ sind stückweise polynomiale Funktionen auf der reellen Achse. Normalerweise ist nur das Intervall $[t_0, t_{m+1}]$ von Interesse.

(vii) Die Berechnung der Basisfunktionen kann in dem bekannten Dreiecksschema erfolgen.

(viii) Die N_i^0 liefern auf dem Intervall $[t_0, t_{m+1})$ eine Zerlegung der Eins und sind positiv.

Wie in vorstehender Bemerkung erwähnt, müssen die Knoten nicht verschieden sein. Ähnlich wie in Beispiel 4.4.3 definieren wir zu einer Knotenfolge $\mathcal{T}$ eine erweiterte Knotenfolge $\tilde{\mathcal{T}}$, bei der $\mathcal{T}$ durch Hilfsknoten an den Intervallenden ergänzt werden. Allerdings fallen hier die Hilfsknoten mit den Randknoten zusammen:

Definition 4.4.6 (Erweiterte Knotenfolge) *Zu einer Knotenfolge $\mathcal{T} = \{x_0, \ldots, x_n\}$ mit $n+1$ paarweise verschiedenen Knoten nennen wir die Knotenfolge $\tilde{\mathcal{T}} = \{t_1, \ldots, t_{n+2k}\}$ mit $n + 2k = \ell + k = m$ Knoten, welche dadurch erzeugt wird, dass die Randknoten k-fach gezählt werden und die Knoten wie folgt benannt werden*

$$\begin{array}{rcccccl} a = & x_0 & < & x_1 & < & \cdots < & x_n = b \\ t_1 = \cdots = & t_k & < & t_{k+1} & < & \cdots < & t_{n+k+1} = \cdots = t_{n+2k} \end{array}$$

*eine **erweiterte Knotenfolge** .*

Bemerkung 4.4.7 (Nummerierung der erweiterten Knotenfolge) *Natürlich ist auch eine andere Nummerierung der erweiterten Knotenfolge möglich. Vergleiche hierzu Beispiel 4.4.3.*

Bemerkung 4.4.8 (Begründung für das Einführen einer erweiteten Knotenfolge) *Zunächst stellt sich die Frage, warum wir überhaupt eine erweiterte Knotenfolge benötigen. Mit $n+1$ paarweise verschiedenen Knoten können $n-k$ unabhängige B-Splines vom Grad k konstruiert werden, wobei bezogen auf die Kardinalität $\ell = n + k$ des Splineraums noch $2k$ Freiheitsgrade frei sind. Durch das Einführen einer erweiterten Knotenfolge wird diese Lücke geschlossen.*

Die rekursive Definition der B-Spline-Basisfunktion (4.32), (4.33) liefert einen einfachen Algorihmus, um die Werte der B-Spline-Basisfunktionen zu einem gegebenen $x \in [t_i, t_{i+1})$ und einer gegebenen erweiterten Knotenfolge $\tilde{\mathcal{T}}$ zu bestimmen. Hierzu betrachten wir zwei Beispiele:

Beispiel 4.4.9 (Rekursive Berechnung der B-Spline-Basisfunktionen) Es sei $\tilde{\mathcal{T}} = \{t_1 = 0, t_2 = 0, t_3 = 0, t_4 = 1, t_5 = 1, t_6 = 1\}$ und $k = 2$. Die B-Spline-Basisfunktionen vom Grade 0, 1 und 2 zu dieser Knotenfolge lauten dann:

Stückweise konstante B-Spline-Basisfunktionen:

$$\begin{aligned} N_0^0(x) &= N_1^0(t) = 0 \quad -\infty < x < \infty \\ N_2^0(x) &= \begin{cases} 1 & 0 \le x < 1 \\ 0 & \text{sonst} \end{cases} \\ N_3^0(x) &= N_4^0(x) = 0 \quad -\infty < x < \infty \end{aligned}$$

Stückweise lineare B-Spline-Basisfunktionen:

$$\begin{aligned} N_0^1(x) &= \frac{x-0}{0-0}N_0^0(x) + \frac{0-x}{0-0}N_1^0(x) = 0 && -\infty < x < \infty \\ N_1^1(x) &= \frac{x-0}{0-0}N_1^0(x) + \frac{1-x}{1-0}N_2^0(x) = \begin{cases} 1-x & 0 \le x < 1 \\ 0 & \text{sonst} \end{cases} \\ N_2^1(x) &= \frac{x-0}{1-0}N_2^0 + \frac{1-x}{1-1}N_3^0(x) = \begin{cases} x & 0 \le x < 1 \\ 0 & \text{sonst} \end{cases} \\ N_3^1(x) &= \frac{x-1}{1-1}N_3^0 + \frac{1-x}{1-1}N_4^0(x) = 0 && -\infty < x < \infty \end{aligned}$$

Stückweise quadratische B-Spline-Basisfunktionen:

$$
\begin{aligned}
N_0^2(x) &= \frac{x-0}{0-0}N_0^1 + \frac{1-x}{1-0}N_1^1(x) &&= \begin{cases}(1-x)^2 & 0 \le x < 1\\ 0 & \text{sonst}\end{cases}\\
N_1^2(x) &= \frac{x-0}{1-0}N_1^1 + \frac{1-x}{1-0}N_2^1(x) &&= \begin{cases}2x(1-x) & 0 \le x < 1\\ 0 & \text{sonst}\end{cases}\\
N_2^2(x) &= \frac{x-0}{1-0}N_2^1 + \frac{1-x}{1-1}N_3^1(x) &&= \begin{cases}x^2 & 0 \le x < 1\\ 0 & \text{sonst}\end{cases}
\end{aligned}
$$

Bemerkung 4.4.10 (B-Splines-Darstellung als Verallgemeinerung der Bézier-Darstellung)
Man beachte, dass N_i^2 auf das Intervall $[0,1)$ restringiert gerade die quadratischen Bernstein-Polynome sind. Aus diesem Grunde ist die B-Spline-Darstellung mit einem Knotenvektor der Form

$$U = \{\ \underbrace{0,\ldots,0}_{(k+1)-mal}\ ,\ \underbrace{1,\ldots,1}_{(k+1)-mal}\ \}$$

eine Verallgemeinerung der Bézier-Darstellung ist.

Beispiel 4.4.11 Es sei $\tilde{\mathcal{T}} = \{t_1 = t_2 = t_3 = 0, t_4 = 1, t_5 = 2, t_6 = 3, t_7 = t_8 = 4, t_9 = t_{10} = t_{11} = 5\}$ und $k = 2$.
Die B-Spline-Basisfunktionen vom Grade 0, 1 und 2 lauten dann:

Stückweise konstanten B-Spline-Basisfunktionen:

$$
\begin{aligned}
N_0^0(x) &= N_1^0(x) = 0 \quad -\infty < x < \infty\\
N_2^0(x) &= \begin{cases}1 & 0 \le x < 1\\ 0 & \text{sonst}\end{cases}\\
N_3^0(x) &= \begin{cases}1 & 1 \le x < 2\\ 0 & \text{sonst}\end{cases}\\
N_4^0(x) &= \begin{cases}1 & 2 \le x < 3\\ 0 & \text{sonst}\end{cases}\\
N_5^0(x) &= \begin{cases}1 & 3 \le x < 4\\ 0 & \text{sonst}\end{cases}\\
N_6^0(x) &= 0 \quad -\infty < x < \infty\\
N_7^0(x) &= \begin{cases}1 & 4 \le x < 5\\ 0 & \text{sonst}\end{cases}\\
N_8^0(x) &= N_9^0(x) = 0 \quad -\infty < x < \infty
\end{aligned}
$$

Stückweise lineare B-Spline-Basisfunktionen:

$$
\begin{aligned}
N_0^1(x) &= \frac{x-0}{0-0}N_0^0(x) + \frac{0-x}{0-0}N_1^0(x) &&= 0 && -\infty < x < \infty\\
N_1^1(x) &= \frac{x-0}{0-0}N_1^0(x) + \frac{1-x}{1-0}N_2^0(x) &&= \begin{cases}1-x & 0 \le x < 1\\ 0 & \text{sonst}\end{cases}\\
N_2^1(x) &= \frac{x-0}{1-0}N_2^0(x) + \frac{2-x}{2-1}N_3^0(x) &&= \begin{cases}x & 0 \le x < 1\\ 2-x & 1 \le x < 2\\ 0 & \text{sonst}\end{cases}
\end{aligned}
$$

$$
\begin{aligned}
N_3^1(x) &= \frac{x-1}{2-1}N_3^0(x) + \frac{3-x}{3-2}N_4^0(x) &&= \begin{cases} x-1 & 1 \le x < 2 \\ 3-x & 2 \le x < 3 \\ 0 & \text{sonst} \end{cases} \\
N_4^1(x) &= \frac{x-2}{3-2}N_4^0(x) + \frac{4-x}{4-3}N_5^0(x) &&= \begin{cases} x-2 & 2 \le x < 3 \\ 4-x & 3 \le x < 4 \\ 0 & \text{sonst} \end{cases} \\
N_5^1(x) &= \frac{x-3}{4-3}N_5^0(x) + \frac{4-x}{4-4}N_6^0(x) &&= \begin{cases} x-3 & 3 \le x < 4 \\ 0 & \text{sonst} \end{cases} \\
N_6^1(x) &= \frac{x-4}{4-4}N_6^0(x) + \frac{5-x}{5-4}N_7^0(x) &&= \begin{cases} 5-x & 4 \le x < 5 \\ 0 & \text{sonst} \end{cases} \\
N_7^1(x) &= \frac{x-4}{5-4}N_7^0(x) + \frac{5-x}{5-5}N_8^0(x) &&= \begin{cases} x-4 & 4 \le x < 5 \\ 0 & \text{sonst} \end{cases} \\
N_8^1(x) &= \frac{x-5}{5-5}N_8^0(x) + \frac{5-x}{5-5}N_9^0(x) &&= 0 \qquad -\infty < x < \infty
\end{aligned}
$$

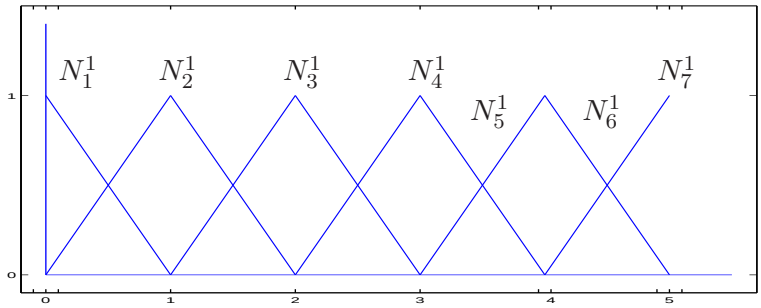


Abb. 4.18: Die stückweise linearen Basisfunktionen zu $\tilde{T} = \{0, 0, 0, 1, 2, 3, 4, 4, 5, 5, 5\}$.

Stückweise quadratische B-Spline-Basisfunktionen: Die folgenden N_i^2 sind bis auf die angegebenen Intervalle jeweils Null.

$$
\begin{aligned}
N_0^2(x) &= \frac{x-0}{0-0}N_0^1(x) + \frac{1-x}{1-0}N_1^1(x) &&= (1-x)^2 && 0 \le x < 1 \\
N_1^2(x) &= \frac{x-0}{1-0}N_1^1(x) + \frac{2-x}{2-0}N_2^1(x) &&= \begin{cases} 2x - \frac{3}{2}x^2 & 0 \le x < 1 \\ \frac{1}{2}(2-x)^2 & 1 \le x < 2 \end{cases} \\
N_2^2(x) &= \frac{x-0}{2-0}N_2^1(x) + \frac{3-t}{3-1}N_3^1(x) &&= \begin{cases} \frac{1}{2}x^2 & 0 \le x < 1 \\ -\frac{3}{2} + 3x - x^2 & 1 \le x < 2 \\ \frac{1}{2}(3-x)^2 & 2 \le x < 3 \end{cases} \\
N_3^2(x) &= \frac{x-1}{3-1}N_3^1(x) + \frac{4-x}{4-2}N_4^1(x) &&= \begin{cases} \frac{1}{2}(x-1)^2 & 1 \le x < 2 \\ -\frac{11}{2} + 5x - x^2 & 2 \le x < 3 \\ \frac{1}{2}(4-x)^2 & 3 \le x < 4 \end{cases} \\
N_4^2(x) &= \frac{x-2}{4-2}N_4^1(x) + \frac{4-x}{4-3}N_5^1(x) &&= \begin{cases} \frac{1}{2}(x-2)^2 & 2 \le x < 3 \\ -16 + 10x - \frac{3}{2}x^2 & 3 \le x < 4 \end{cases} \\
N_5^2(x) &= \frac{x-3}{4-3}N_5^1(x) + \frac{5-x}{5-4}N_6^1(x) &&= \begin{cases} (x-3)^2 & 3 \le x < 4 \\ (5-x)^2 & 4 \le x < 5 \end{cases} \\
N_6^1(x) &= \frac{x-4}{5-4}N_6^1(x) + \frac{5-x}{5-4}N_7^1(x) &&= 2(x-4)(5-x) && 4 \le x < 5 \\
N_7^1(x) &= \frac{x-4}{5-4}N_7^1(x) + \frac{5-x}{5-5}N_8^1(x) &&= (x-4)^2 && 4 \le x < 5
\end{aligned}
$$

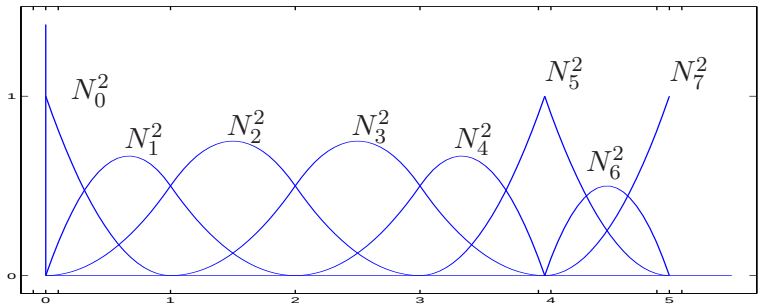


Abb. 4.19: Die stückweise quadratischen Basisfunktionen zu $\tilde{\mathcal{T}} = \{0, 0, 0, 1, 2, 3, 4, 4, 5, 5, 5\}$.

In Abbildung 4.20 ist die Zusammensetzung von N_3^2 aus den jeweiligen stückweise polynomialen Funktionen auf den einzelnen Teilintervallen grafisch dargestellt.

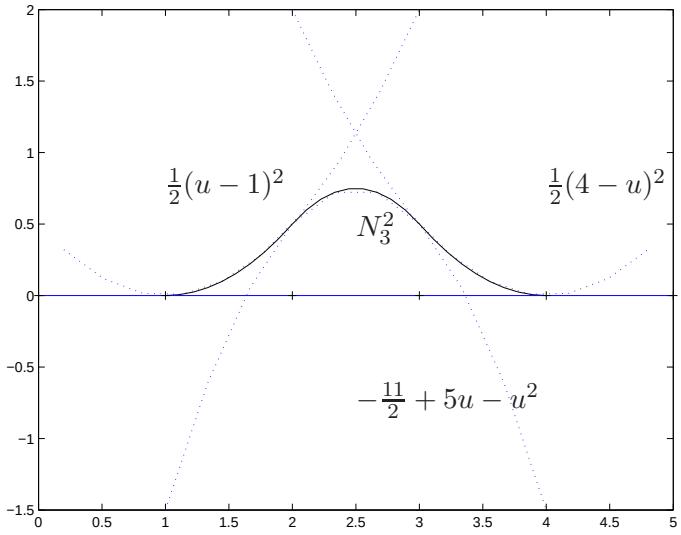


Abb. 4.20: Die Zerlegung von N_3^2 in seine stückweise polynomialen Teilfunktionen.

Es ist wichtig, den Effekt von mehrfachen Knoten zu verstehen. Man betrachte die Funktionen N_0^2, N_1^2, N_2^2, N_5^2 und N_6^2 in Abbildung 4.19. Beachtet man die rekursive Definition der Basisfunktionen (4.33), so stellt man fest, dass sie jeweils nur von 4 Knoten abhängen, nämlich:

$$\begin{aligned} N_0^2 &: \{0, 0, 0, 1\} \\ N_1^2 &: \{0, 0, 1, 2\} \\ N_2^2 &: \{0, 1, 2, 3\} \\ N_5^2 &: \{3, 4, 4, 5\} \\ N_6^2 &: \{4, 4, 5, 5\} \end{aligned}$$

Der Begriff **Vielfachheit eines Knotens**, kann man verstehen

- in Bezug auf eine Knoten im Knotenvektor oder
- in Bezug auf einen Knoten bezüglich einer Basisfunktion.

Zum Beispiel hat $t = 0$ die Vielfachheit 3 im o.g. Knotenvektor $\tilde{\mathcal{T}}$, aber in Bezug auf die Basisfunktion N_1^2 ist $t = 0$ ein Knoten mit der Vielfachheit 2. Die Basisfunktionen sind stückweise

polynomiale Funktionen, d.h. im Inneren der Intervallen (t_j, t_{j+1}) sind sie beliebig glatt. Unstetigkeiten können also nur an den Knoten auftreten. Für $t = 0$ stellt man fest, dass N_0^2 unstetig ist, N_1^2 C^0 stetig ist, N_2^2 C^1 stetig ist und N_5^2 und all seine Ableitungen dort Null von beiden Seiten ist. N_1^2 sieht $t = 0$ als doppelten Knoten, N_2^2 sieht $t = 0$ als einfachen Knoten und N_5^2 enthält $t = 0$ gar nicht als Knoten.

Das Ziel ist es ja, eine *stabile* Basis von $\mathcal{S}^k(\mathcal{T})$ zu konstruieren. Betrachte nun

$$\mathbb{B}^k(\tilde{\mathcal{T}}) := \text{span}\{N_j^k\,; j = 0, \ldots, \ell - 1\}$$

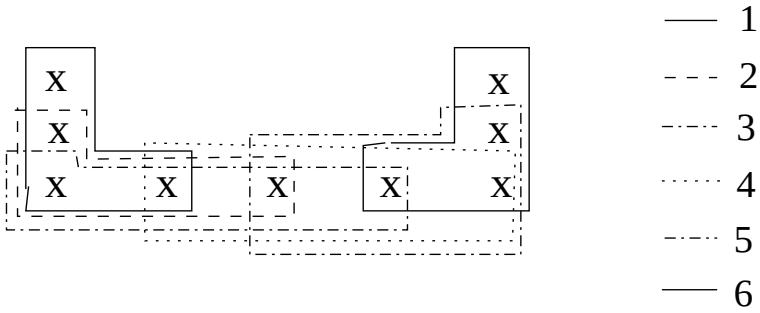
zu einer erweiterten Knotenfolge $\tilde{\mathcal{T}}$. Um zu zeigen, dass die durch Definition 4.4.4 erklärten Funktionen dies liefern und somit ihre Bezeichnung als Basisfunktionen gerechtfertigt ist, wollen wir später den folgenden Satz beweisen:

Satz 4.4.12 (Basis des Splineraums) *Sei $\mathcal{T} = \{x_0, \ldots, x_n\}$ eine Knotenfolge paarweise verschiedener Knoten und $\tilde{\mathcal{T}} = \{t_1, \ldots, t_{n+2k}\}$. Dann gilt*

$$\mathcal{S}^k(\mathcal{T}) = \mathbb{B}^k(\tilde{\mathcal{T}}),$$

d.h. die B-Spline-Basisfunktionen bilden eine Basis des Splineraums.

Beispiel 4.4.13 Sei $\mathcal{T} = \{x_0, \ldots, x_4\}$, $x_0 = a$, $x_4 = b$, d.h. $n = 4$, $\dim S^2(\mathcal{T}) = 2 + n = 6$



Wenn Satz 4.4.12 bewiesen ist, gilt für jeden Spline $s \in \mathcal{S}^k(\mathcal{T})$ die **Darstellung als Linearkombination**

$$s(x) = \sum_{j=0}^{\ell-1} c_j N_j^k(x), \tag{4.34}$$

d.h. s lässt sich durch $\boldsymbol{c} = (c_0, \ldots, c_{\ell-1})^T \in \mathbb{R}^\ell$ codieren. Angenommen, wir wollen $s(x)$ (oder $s'(x)$) für ein $x \in [a, b]$ berechnen. Man könnte

- mit den obigen Rekursionen $N_j^k(x)$ für jedes j bestimmen,
- dann mit c_j multiplizieren und aufsummieren.

Man kann aber eine deutlich **effizientere Rekursion** für die c_j angeben:

$$\begin{aligned} s(x) &= \sum_{j=0}^{\ell-1} c_j \left\{ \frac{x - t_j}{t_{j+k} - t_j} N_j^{k-1}(x) + \frac{t_{j+k+1} - x}{t_{j+k+1} - t_{j+1}} N_{j+1}^{k-1}(x) \right\} \\ &= \sum_{j=1}^{\ell-1} \underbrace{\left(c_j \frac{x - t_j}{t_{j+k} - t_j} + c_{j-1} \frac{t_{j+k} - x}{t_{j+k} - t_j} \right)}_{=: c_j^{[1]}} N_j^{k-1}(x) \end{aligned}$$

da $N_0^{k-1}(x) \equiv N_{\ell-1}^{k-1}(x) \equiv 0$ für alle $x \in (a, b)$ im Beispiel 4.4.13.

Dieses Argument wiederholt man und erhält für $\nu < k$

$$s(x) = \sum_{j=\nu}^{\ell-1} c_j^{[\nu]} N_j^{k-\nu}(x) \tag{4.35}$$

mit

$$c_j^{[\nu]} := \begin{cases} c_j, & \text{für } \nu = 0, \\ \frac{x-t_j}{t_{j+k-(\nu-1)}-t_j} c_j^{[\nu-1]} + \frac{t_{j+k-(\nu-1)}-x}{t_{j+k-(\nu-1)}-t_j} c_{j-1}^{[\nu-1]}, & \text{sonst.} \end{cases} \tag{4.36}$$

Sei nun $x \in [t_i, t_{i+1})$ für ein m (dieses kann man leicht bestimmen). Wegen

$$N_j^0(x) \equiv \chi_{[t_j, t_{j+1})}(x)$$

, d.h. wegen der Lokalität der B-Splines ergibt sich für $\nu = k$

$$s(x) = \sum_{j=k}^{n} c_j^{[k]} N_j^0(x) = c_i^{[k]} \quad \text{für alle } x \in [t_i, t_{i+1}). \tag{4.37}$$

Natürlich hängen die $c_j^{[\nu]}$ von x ab: $c_j^{[\nu]} = c_j^{[\nu]}(x)$. Damit erhalten wir — unter der Annahme, dass Satz 4.4.12 gilt:

Algorithmus 4.4.1 (Rekursive Berechnung von $s(x)$) *Gegeben sei eine erweiterte Knotenfolge $\tilde{\mathcal{T}}$.*

1.) *Für $x \in [a, b]$ bestimme i, so dass $k \le i \le \ell$ mit $x \in [t_i, t_{i+1})$.*

2.) *Setze $c_j^{[0]} := c_j,\ j = 0, \ldots, \ell - 1$.*

3.) *Für $\nu = 1, \ldots, k-1$, berechne*

$$c_j^{[n]} = c_j^{[n]}(x), \quad j = i - k + \nu, \ldots, i,$$

nach (4.36).

4.) *Setze $s(x) := c_j^{[k-1]}(x)$.*

Bemerkung 4.4.14 (Aufwand der algorithmischen Berechnung von $s(x)$) *(a) Algorithmus 4.4.1 benötigt etwa genauso viele Operationen wie (4.33) zur Berechnung* eines N_j^k*!*

(b) Offensichtlich ist Algorithmus 4.4.1 analog zum Neville-Aitken-*Schema zur Auswertung der Polynominterpolation.*

Satz 4.4.15 (Ableitung von B-Splines) *Ist t_ν ein j-facher Knoten, d.h.*

$$t_{\nu-1} < t_\nu = \cdots = t_{\nu+j-1} < t_{\nu+j},$$

so ist N_i^k an der Stelle t_ℓ mindestens $(k-j)$-mal stetig differenzierbar. Für die Ableitung von N_i^k gilt

$$\frac{d}{dt} N_i^k(t) = (k-1) \cdot \left(\frac{N_i^{k-1}(t)}{t_{i+k} - t_i} - \frac{N_{i+k}^{k-1}(t)}{t_{i+k+1} - t_{i+1}} \right)$$

Beweis. Siehe Deuflhard S. 245 Korollar 7.52 □

Bemerkung 4.4.16 (Ableitungen von Splines) *Wegen Satz 4.4.15 erhält man ein analoges Schema für die Ableitungen*

$$s^{(d)}(x) = (k-1)\cdots(k-d)\sum_{j=1+d}^{n} c_j^{(d)} N_j^{k-d}(x) \tag{4.38}$$

mit

$$c_j^{(d)} = \begin{cases} c_j, & d = 0, \\ \frac{c_j^{(d-1)} - c_{j-1}^{(d-1)}}{t_{j+k-d} - t_j}, & d > 0. \end{cases} \tag{4.39}$$

Bemerkung 4.4.17 (Weitere Eigenschaften der B-Splines) *(a) B-Splines bilden eine* „Partition der Eins“

$$\sum_{j=0}^{\ell-1} N_j^k(x) = 1,$$

für alle $x \in [a,b]$. Denn wähle $c_j = 1$ in (4.34), d.h. $c_j^{[0]} = 1$ für alle j, dann folgt induktiv

$$c_j^{[\nu]} = \frac{x - t_j}{t_{j+k-(\nu-1)} - t_j} \underbrace{c_j^{[\nu-1]}}_{=1} + \frac{t_{j+k-(\nu-1)} - x}{t_{j+k-(\nu-1)} - t_j} \underbrace{c_{j-1}^{[\nu]}}_{=1} = 1.$$

(b) Per Induktion nach k zeigt man allgemeiner die so genannte **Marsdens**[5] ***Identiät***

$$(x-y)^{k-1} = \sum_{j=0}^{\ell-1} \prod_{i=1}^{k-1} (t_{j+i} - y) N_j^k(x), \quad k \geq 1, \tag{4.40}$$

*(*Übung*). Differenziert man beide Seiten nach y und wertet dies für $y = 0$ aus, erhält man*

$$x^m = \sum_{j=0}^{\ell-1} \left(\frac{(-1)^{k-1-m}}{(k-1)\cdots(m+1)} \Psi_{j,k}^{(k-1-m)}(0) \right) N_j^k(x), \tag{4.41}$$

mit $0, \ldots, k-1$ und

$$\Psi_{j,k}(y) := \prod_{i=1}^{k-1} (t_{j+i} - y). \tag{4.42}$$

Beweis von Satz 4.4.12: $N_j^k \in S^k(\mathcal{T})$ impliziert $S^k(\mathcal{T}) \subseteq \mathbb{B}^k(\tilde{\mathcal{T}})$.
Auf jedem Teilintervall $[t_i, t_{i+1})$ leben genau k B-Splines

$$N_{i-k+1}^k, \ldots, N_i^k.$$

Wegen (4.41) kann jedes Polynom als Linearkombination der N_j^k dargestellt werden. Daraus folgt, die B-Splines sind linear unabhängig, also

$$\begin{aligned} \dim\{N_j^k : j = 1, \ldots, n+k = n\} = \ell &= \dim \mathbb{B}^k(\tilde{\mathcal{T}}) \\ &= \dim S^k(\mathcal{T}), \end{aligned}$$

was die Behauptung zeigt. □

Bemerkung 4.4.18 *(a) Ein wichtiger Grund für den Erfolg von Splines ist die* Stabilität *der B-Spline-Basis: Für alle $k \in \mathbb{N}$ existieren Konstanten $0 < c < C < \infty$, so dass für alle $\tilde{\mathcal{T}} = \{t_j\}_{j=1}^{\ell+k}$, $\boldsymbol{d} = (d_0, \ldots, d_{\ell-1})^T$ gilt*

$$c\|\boldsymbol{d}\|_\infty \leq \max_{x\in[a,b]} \left| \sum_{j=0}^{\ell-1} d_j N_j^k(x) \right| \leq C\|\boldsymbol{d}\|_\infty \tag{4.43}$$

mit $\|\boldsymbol{d}\|_\infty := \max\limits_{j=0,\ldots,\ell-1} |d_j|$.

(b) Für die Approximationsgüte *gilt: für alle $k \in \mathbb{N}$ existiert $0 < c < \infty$, so dass für alle $m \leq k$ und $f \in C^m$ gilt*

$$\inf_{S_k \in S^k(\mathcal{T})} \|f - S_k\|_\infty \leq c\, h^m \|f^{(m)}\|_\infty$$

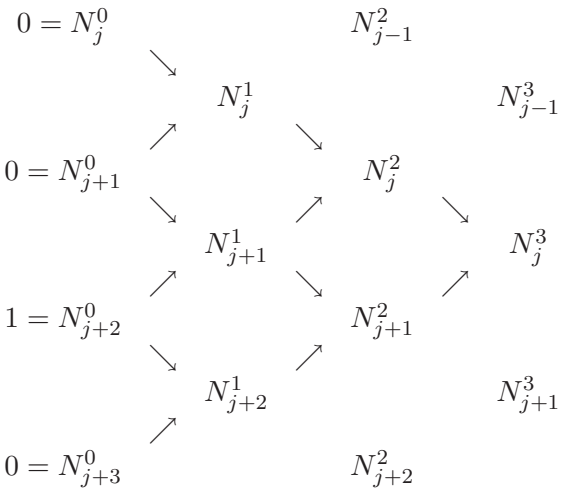
mit $h := \max\limits_{j=0,\ldots,\ell-1} (t_{j+1} - t_j)$ und $\|g\|_\infty := \sup\limits_{x\in[a,b]} |g(x)|$, vgl. Satz 4.0.2

(c) Das Spline-Interpolationsproblem ist eindeutig lösbar genau dann, wenn in dem Träger jedes B-Splines mindestens eine Stützstelle fällt.

(d) Numerisch erfordert dies die Lösung eines linearen Gleichungssystems mit einer Tridiagonalmatrix *($\mathcal{O}(n)$!).*

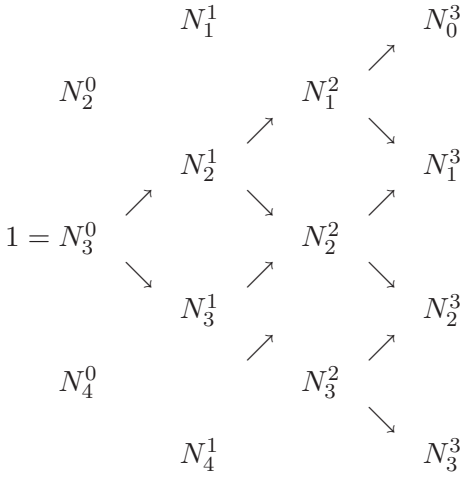
4.4.2 Effiziente Auswertung der B-Spline-Basisfunktionen

Die Funktion N_j^3 ist eine Linearkombination der Funktionen N_j^0, N_{j+1}^0, N_{j+2}^0 und N_{j+1}^0. Somit ist N_j^3 nur von Null verschieden für $t \in [t_j, t_{j+4})$.



Tab. 4.3: N_j^3 ist nur auf dem Intervall $[t_j, t_{j+4})$ von Null verschieden.

In jedem Knotenintervall $[t_j, t_{j+1})$ sind maximal $k+1$ der N_i^k von Null verschieden, nämlich $N_{j-k}^k, \ldots, N_j^k$. Auf $[t_3, t_4)$ ist z.B. N_3^0 die einzige nichtverschwindende Basisfunktion vom Grad Null. Somit sind $N_0^3, \ldots, N_3^3$ die einzigen von Null verschiedenen kubischen Funktionen auf $[t_3, t_4)$. Diese Eigenschaft ist in der folgenden Tabelle 4.4 dargestellt.



Tab. 4.4: N_3^0 ist nur auf dem Intervall $[t_3, t_4)$ von Null veschieden, Somit sind auch $N_0^3, \ldots, N_3^3$ die einzigen von Null verschiedenen kubischen Funktionen auf $[t_3, t_4)$.

4.4.2.1 Ableitung der B-Splines

MATLAB-Funktion: BasisFunc.m

```
1 function N = BasisFunc(i,p,U,t)
```

```
% compute the nonvanishing basis functions
N = zeros(p+1,1);
N(1) = 1;
for j=1:p
  left(j) = t - U(i+1-j);
  right(j) = U(i+j) - t;
  saved = 0;
  for r=1:j
    temp = N(r)/(right(r)+left(j-r+1));
    N(r) = saved + right(r) * temp;
    saved = left(j-r+1) * temp;
  end
  N(j+1) = saved;
end
```

MATLAB-Funktion: FindSpan.m

```
function mid = FindSpan(p,U,t)
% returns the knot span index
n = length(U)-p;
if t==U(end)          % special case
  mid = n-1;
  return
end
low = p;
high = n;
mid = floor((low+high)/2);
while t<U(mid) | t>= U(mid+1)
  if t<U(mid)
    high = mid;
  else
    low = mid;
  end
  mid = floor((low+high)/2);
end
```

MATLAB-Funktion: CurvePoint.m

```
function value = CurvePoint(p,U,P,t)
% compute point on B-spline curve
span = FindSpan(p,U,t);
B = BasisFunc(span,p,U,t);
value = 0*P(:,1);
for i=0:p
  value = value + B(i+1) * P(:,span-p+i);
end
```

4.4.2.2 Ableitung der B-Splines

MATLAB-Funktion: AllBasisFunc.m

```
function N = AllBasisFunc(i,p,U,t)
% compute the nonvanishing basis functions
N = zeros(p+1,p+1);
N(1,1) = 1;
for j=1:p
  left(j) = t - U(i+1-j);
  right(j) = U(i+j) - t;
  saved = 0;
  for r=1:j
    temp = N(r,j)/(right(r)+left(j-r+1));
    N(r,j+1) = saved + right(r) * temp;
    saved = left(j-r+1) * temp;
  end
  N(j+1,j+1) = saved;
end
```

MATLAB-Funktion: CurveDerivPts.m

```
function PK = CurveDerivPts(p,U,P,d,r1,r2)
% compute control points of curve derivatives
r = r2-r1;
for i=1:r+1
  PK(:,1,i)=P(:,r1+i);
end
for k=2:d+1
  tmp = p-k+2;
  for i=1:r-k+2
    PK(:,k,i)=tmp*(PK(:,k-1,i+1)-PK(:,k-1,i))/(U(r1+i+p+1)-U(r1+i+k
       -1));
  end
end
```

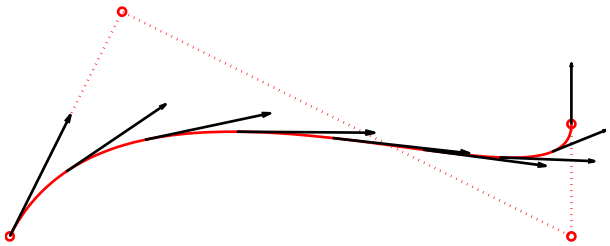


Abb. 4.21: B-Spline mit Ableitungen.

MATLAB-Funktion: CurveDerivs.m

```
function CK = CurveDerivs(p,U,P,t,d)
% Compute curve derivatives
du = min(d,p);
CK(1:size(P,1),[p+2:d+1]) = 0;
span = FindSpan(p,U,t);
N = AllBasisFunc(span,p,U,t);

PK = CurveDerivPts(p,U,P,du,span-p-1,span-1);

for k=1:du+1
 CK(:,k) = 0;
 for j=1:p-k+2
   CK(:,k) = CK(:,k) + N(j,p-k+2)*PK(:,k,j);
 end
end
```

B-Spline mit $k = 3$ mit Ableitungen

MATLAB-Beispiel:

```
>> P =[0,1,5,5;
       0,2,0,1];
>> U = [0,0,0,0,1,1,1,1];  k=3;
>> s = linspace(U(1),U(end),201);
>> for i = 1:length(s)
     C(:,k) = CurvePoint(p,U,P,s(i));
   end
>> plot(C(1,:),C(2,:),'-', ...
        P(1,:),P(2,:),'o:');
>> hold on
>> for j = 1 :25:length(s)
     V = CurveDerivs(k,U,P,s(j),1);
     quiver(V(1,1),V(2,1), ...
            0.2*V(1,2),0.2*V(2,2))
   end
```

4.5 Rationale B-Splines

MATLAB-Funktion: RatCurvePoint.m

```
function value = RatCurvePoint(p,U,Pw,t)
% compute point on B-spline curve
span = FindSpan(p,U,t);
B = BasisFunc(span,p,U,t);
value = 0*Pw(:,1);
for i=0:p
  value = value + B(i+1) * Pw(:,span-p+i);
end
value = value(1:end-1)/value(end);
```

MATLAB-Funktion: RatCurveDerivs.m

```
function CK = RatCurveDerivs(p,U,Pw,t,d)
% compute derivatives on rational B-spline curve
du = min(d,p);

ders = CurveDerivs(p,U,Pw,t,d);
Aders = ders(1:end-1,:);
wders = ders(end,:);

CK(1:size(Aders,1),p+2:d+1) = 0;

for k=1:du+1
   v = Aders(:,k);
   for i=1:k-1
     v = v - nchoosek(k-1,i)*wders(i+1)*CK(:,k-i);
   end
   CK(:,k) = v/wders(1);
end
```

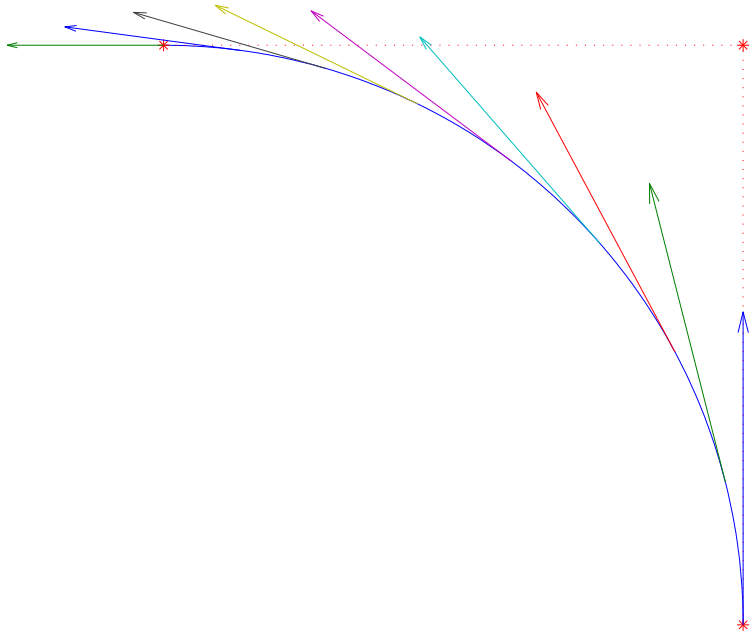


Abb. 4.22: Rationaler B-Spline mit Ableitung.

Rationaler B-Spline mit Ableitungen

MATLAB-Beispiel:

```
>> w = [1, 1,  2];
>> P = [1, 1,  0;
        0  1,  1];
>> U = [0,0,0,1,1,1];   k=2;
>> s = linspace(0,1,201);
>> Pw = [P(1,:).*w;P(2,:).*w;w];
>> for i = 1:length(s)
     Cw(:,k) = RatCurvePoint(p,U,Pw,s(i));
   end
>> plot(Cw(1,:),Cw(2,:),'-', ...
        P(1,:),P(2,:),'*:');
>> hold on
>> for j = 1 :25:length(s)
     CK = RatCurveDerivs(k,U,Pw,s(j),1);
      quiver(CK(1,1),CK(2,1),0.3*CK(1,2)
         ,0.3*CK(2,2))
   end
```

A Lineare Differenzengleichung

Wir beschränken uns hier auf lineare Differenzengleichungen mit konstanten Koeffizienten.

Definition A.0.1 *Sei $m \in \mathbb{N}$. Eine reelle Folge (a_n) ist durch $a_0, a_1, \ldots, a_{m-1}$ und $\mu_0, \mu_1, \ldots, \mu_{m-1}$ $(\mu_0 \neq 0)$ mit der Rekursionsvorschrift*

$$a_{n+m} = \sum_{i=0}^{m-1} \mu_i a_{n+i} \qquad \text{für alle } n \in \mathbb{N}_0 \tag{A.1}$$

eindeutig bestimmt. Diese Rekursionsvorschrift heißt lineare Differenzengleichung m-ter Ordnung.

Definiert man

$$\hat{a}_k := \begin{pmatrix} a_k \\ \vdots \\ a_{k+m-1} \end{pmatrix} \quad \text{und} \quad A := \begin{pmatrix} 0 & 1 & 0 & \ldots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & 0 \\ 0 & \ldots & \ldots & 0 & 1 \\ \mu_0 & \mu_1 & \ldots & \ldots & \mu_{m-1} \end{pmatrix} \tag{A.2}$$

so folgt $\hat{a}_{k+1} = A\hat{a}_k$, also gilt $\hat{a}_n = A^n \hat{a}_0$.

Definition A.0.2 *Die Eigenwerte $\lambda_1, \ldots, \lambda_m$ von A aus* (A.2) *heißen auch Eigenwerte der Differenzengleichung aus* (A.1)*.*

Wir beschränken uns zunächst auf den Fall, dass die Eigenwerte $\lambda_1, \ldots, \lambda_m$ von A paarweise verschieden sind, d. h. $A = B^{-1}DB$ ist diagonalisierbar mit der Diagonalmatrix

$$D = \begin{pmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_m \end{pmatrix}.$$

Damit gibt es Vektoren $b^{(1)}, \ldots, b^{(m)} \in \mathbb{R}^m$ mit $\hat{a}_n = B^{-1}D^n B\hat{a}_0 = \lambda_1^n b^{(1)} + \ldots + \lambda_m^n b^{(m)}$. Betrachtet man die erste Zeile, so gibt es Konstanten $c_1, \ldots, c_m$ mit

$$a_n = c_1\lambda_1^n + \ldots + c_n\lambda_m^n\,.$$

Mit Hilfe der Anfangswerte $a_0, \ldots, a_{m-1}$ berechnet man die Koeffizienten $c_1, \ldots, c_n$.

Satz A.0.3 *Seien die Nullstellen $x_1, \ldots, x_m \in \mathbb{C}$ des Polynoms*

$$p(x) = x^m - \sum_{k=0}^{m-1} \mu_k x^k$$

paarweise verschieden, so ist $(x_1^n), \ldots, (x_m^n)$ eine Basis des Lösungsraums der linearen Differenzengleichung (A.1)*, d. h. a_n ist genau dann eine Lösung von* (A.1)*, wenn es Konstanten $c_1, \ldots, c_n \in \mathbb{R}$ gibt mit $a_n = c_1x_1^n + \ldots + c_mx_m^n$*

Beweis. Wir zeigen mit vollständiger Induktion über m

$$\det(A - \lambda I) = (-1)^m + (-1)^{m-1} \sum_{k=0}^{m-1} \mu_k x^k \,.$$

für A aus (A.2). Für $m = 1$ gilt $\det(A - \lambda I) = -\lambda + \mu_0$. Für $m \geq 2$ gilt nach Auflösung der Determinante nach der ersten Spalte sowie nach Induktionsvoraussetzung

$$\begin{aligned}\det(A - \lambda I) &= -\lambda \left((-1)^{m-1} \lambda^{m-1} + (-1)^m \sum_{k=0}^{m-2} \mu_{k+1} \lambda^k \right) + (-1)^{m-1} \mu_0 \cdot 1 \\ &= (-1)^m \lambda^m + (-1)^{m-1} \sum_{k=0}^{m-1} \mu_k \lambda^k \,.\end{aligned}$$

Damit sind $x_1, \ldots, x_m$ die Eigenwerte von A. Da der Lösungsraum m-dimensional ist, ist $a_n = c_1 x_1^n + \ldots + c_m x_m^n$ für alle $c_1, \ldots, c_n \in \mathbb{R}$ Lösung von (A.1). □

Beispiel A.0.4 Die Fibonacci-Zahlen $(a_n) = (0, 1, 1, 2, 3, 5, \ldots)$ sind durch die Rekursionsformel $a_{n+2} = a_{n+1} + a_n$ und durch die Anfangswerte $a_0 = 0$ und $a_1 = 1$ definiert. Das Differenzengleichungssystem hat dann die Gestalt

$$\begin{pmatrix} a_{n+1} \\ a_{n+2} \end{pmatrix} = A \begin{pmatrix} a_n \\ a_{n+1} \end{pmatrix} \quad \text{mit} \quad A = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} .$$

Für das charakteristische Polynom gilt

$$-x(1-x) - 1 = x^2 - x - 1 = 0 \,.$$

Man erhält zwei unterschiedliche Eigenwerte

$$x_1 = \frac{1 - \sqrt{5}}{2} \quad \text{und} \quad x_2 = \frac{1 + \sqrt{5}}{2} \,,$$

woraus

$$a_n = c_1 \left(\frac{1 - \sqrt{5}}{2} \right)^n + c_2 \left(\frac{1 + \sqrt{5}}{2} \right)^n$$

folgt. Das Einsetzen der Anfangswerte bestimmt

$$c_1 = -\frac{1}{\sqrt{5}} \quad \text{und} \quad c_2 = \frac{1}{\sqrt{5}}$$

und führt zu

$$c_n = \frac{1}{\sqrt{5}} \left(\left(\frac{1 - \sqrt{5}}{2} \right)^n - \left(\frac{1 + \sqrt{5}}{2} \right)^n \right) .$$

Satz A.0.5 *Seien $x_1, \ldots, x_\ell \in \mathbb{C}$ die Nullstellen des Polynoms*

$$p(x) = x^m - \sum_{k=0}^{m-1} \mu_k x^k$$

mit den algebraischen Vielfachheiten $k_1, \ldots, k_\ell \in \mathbb{N}$, so ist a_n genau dann Lösung von (A.1)*, wenn es Polynome $p_i \in \mathbb{P}_{k_i - 1}$ gibt mit $a_n = p_1(n) x_1^n + \ldots + p_\ell(n) x_\ell^n$. Eine Basis des Lösungsraums wäre damit*

$$x_1^n, n x_1^n, \ldots, n^{k_1 - 1} x_1^n, x_2^n, n x_2^n, \ldots, n^{k_2 - 1} x_2^n, \ldots\ldots, x_\ell^n, n x_\ell^n, \ldots, n^{k_\ell - 1} x_\ell^n \,.$$

Beweis. Da mit A aus (A.2) der Rang von $A - x_i I$ gleich $n-1$ ist, gibt es zu jedem Eigenwert x_i einen Jordanblock $J_i \in \mathbb{R}^{k_i \times k_i}$ mit der Gestalt

$$J_i = \begin{pmatrix} x_i & 1 & 0 & \dots & 0 \\ 0 & x_i & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & \ddots & 1 \\ 0 & \dots & \dots & 0 & x_i \end{pmatrix},$$

sodass

$$A = B^{-1} \begin{pmatrix} J_1 & & 0 \\ & \ddots & \\ 0 & & J_n \end{pmatrix} B$$

gilt. Sei

$$E = \begin{pmatrix} 0 & 1 & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & 0 \\ \vdots & & & \ddots & 1 \\ 0 & \dots & \dots & \dots & 0 \end{pmatrix} \in \mathbb{R}^{k_i \times k_i} \quad \text{und} \quad \binom{y}{j} := \frac{(y-j+1)\cdots(y-1)\cdot y}{j!}.$$

Wegen

$$J_i^n = (x_i I + E)^n = \sum_{j=0}^{n} \binom{n}{j} x_i^{n-j} E^j = x_i^n \sum_{j=0}^{k_i-1} \underbrace{\binom{n}{j} x_i^{-j}}_{\in \mathbb{P}_j \text{ bzgl. } n} E^j$$

gibt es vektorwertige Polynome $q^{(i)} \in \mathbb{P}^n_{k_i-1}$ mit

$$\hat{a}_n = B^{-1} \begin{pmatrix} J_1^n & & 0 \\ & \ddots & \\ 0 & & J_\ell^n \end{pmatrix} B\hat{a}_0 = x_1^n q^{(1)}(n) + \ldots + x_\ell^n q^{(\ell)}(n).$$

Da der Lösungsraum m-dimensional ist, ist $a_n = p_1(n)x_1^n + \ldots + p_\ell(n)x_\ell^n$ für alle Polynome $p_i \in \mathbb{P}_{k_i-1}$ Lösung von (A.1). □

Beispiel A.0.6 Ein Beispiel für gleiche Eigenwerte liefert die Rekursionsformel

$$b_{n+2} = 4b_{n+1} - 4b_n,$$

die zum charakteristischen Polynom $x^2 - 4x + 4 = 0$ führt. Mit $x_{1,2} = 2$ erhält man den Ansatz

$$a_n = (c_1 + c_2 n) \cdot 2^n.$$

Wählt man als Anfangswerte $b_0 = 1$ und $b_1 = 4$, so hat die explizite Darstellung die Gestalt $b_n = (1+n) \cdot 2^n$.

Bemerkung A.0.7 *Sollte eine der Eigenwerte x von* (A.1) *komplex sein, so ist auch der komplex konjugierte Eigenwert $\bar{x}$ ebenfalls Eigenwert mit der gleichen algebraischen Vielfachheit. Sei*

$$a_n = c_1 x^n + c_2 \bar{x}^n.$$

Dann folgt $c := c_1 = \bar{c}_2$, d. h. es gilt

$$a_n = 2\Re(c \cdot x^n) = 2|c||x|^n \cos\left(n \arctan \frac{\Im(x)}{\Re(x)} + \arctan \frac{\Im(c)}{\Re(c)}\right).$$

Bemerkung A.0.8 *Eine weitere Möglichkeit Differenzengleichungen zu lösen besteht unter Verwendung der Potenzreihe*

$$f(x) = \sum_{k=0}^{\infty} a_k x^k \,. \tag{A.3}$$

(siehe auch [Meyberg/Vachenauer]). Sei nun $A := \max\{|\mu_i| : i = 0, \ldots, m-1\} > 0$*, dann zeigt man* $|a_k| \leq C(nA)^{k+1-m}$ *mit* $C = \max\{|a_i| : i = 0, \ldots, m-1\}$ *per vollständiger Induktion. Damit existiert der Grenzwert der Potenzreihe* $f(x)$ *für* $|x| < \frac{1}{mA}$*. Geschicktes Aufsummieren*

$$\left(-1 + \sum_{i=1}^{m} \mu_{m-i} x^i\right) f(x) = \sum_{i=0}^{\infty} b_i x^i = \sum_{i=0}^{m-1} b_i x^i$$

eliminiert wegen

$$b_{n+m} = -a_{m+n} + \sum_{i=0}^{m-1} \mu_i a_{n+i} = 0$$

bis auf die ersten $m-1$ *alle weiteren Summanden. Die Koeffizienten der ersten* $m-1$ *Summanden haben die Gestalt*

$$b_k = -a_k + \sum_{i=m-k}^{m-1} \mu_i a_{k-m+i} \,.$$

Umformen von (A.3) *führt zu*

$$f(x) = \frac{b_0 + b_1 x + \ldots + b_{m-1} x^{m-1}}{-1 + \mu_{m-1} x + \mu_{m-2} x^2 + \ldots + \mu_0 x^m} \,.$$

Zerlegt man die rationale Funktion in Partialbrüche der Gestalt

$$\frac{1}{(x-a)^k} = \left(-\frac{1}{a}\right)^k \sum_{i=0}^{\infty} \binom{k+i-1}{k-1} \left(\frac{x}{a}\right)^i \,,$$

so kann diese wiederum als Potenzreihen dargestellt und mit den Koeffizienten der ursprünglichen Potenzreihe verglichen werden.

Bemerkung A.0.9 *Allgemeiner können die Koeffizienten* μ_i *von* n *abhängen, d. h. man erhält die lineare Differenzengleichung*

$$a_{n+m} = \sum_{i=0}^{m-1} \mu_i(n) a_{n+i} \qquad \textit{für alle } n \in \mathbb{N}_0 \,, \tag{A.4}$$

bei der allgemeine Lösungsansätze nicht mehr existieren. Eine Behandlung dieses Themas zeigt [Elayadi]. Anwendungen und rechnergestützte Berechnungen zum Thema Differenzengleichungen findet man u. a. in [Cull].

A.1 Inhomogene lineare Differenzengleichungen

Ersetzen wir (A.1) durch

$$a_{n+m} = \sum_{i=0}^{m-1} \mu_i a_{n+i} + c_n \qquad \text{für alle } n \in \mathbb{N}_0 \,, \tag{A.5}$$

wobei (c_k) eine reelle Folge ist. Seien $(a_k^{(1)})$ und $(a_k^{(2)})$ zwei Lösungen von (A.5), so folgt

$$a_{n+m}^{(2)} - a_{n+m}^{(1)} = \sum_{i=0}^{m-1} \mu_i \left(a_{n+i}^{(2)} - a_{n+i}^{(1)} \right) ,$$

also ist $(a_k^{(2)} - a_k^{(1)})$ Lösung von (A.1). Ist eine sogenannte partikuläre Lösung (a_k) von (A.5) bekannt, so sind alle Lösungen von (A.5) durch Addition der homogenen Lösungen bekannt. Ein allgemeines Verfahren zur Bestimmung einer solchen partikulären Lösung gibt es nicht. Für einige Spezialfälle kann man wie folgt verfahren:

1. Fall: $c := c_n = const.$
Eine partikuläre Lösung wäre die konstante Folge

$$a_n = \frac{c}{1 - \sum_{i=0}^{m-1} \mu_i} .$$

2. Fall: $c_n = p(n)$, mit $p \in \mathbb{P}_r$
Verwenden Sie den Ansatz $a_k = \sum_{i=0}^{r} b_i k^i$, setzen Sie diese in (A.5) ein und vergleichen Sie die Koeffizienten vor den Monomen n^j.

3. Fall: $c_n = u^n$, mit $u \in \mathbb{R}$
Verwenden Sie den Ansatz $a_k = b \cdot u^k$, setzen Sie diese in (A.5) ein und ermitteln Sie b.

4. Fall: $c_n = A_0 \sin(\alpha n) + B_0 \cos(\alpha n)$ mit $A_0, B_0 \in \mathbb{R}$
Verwenden Sie den Ansatz $a_k = A \sin(\alpha k) + B \cos(\alpha k)$, setzen Sie diese in (A.5) ein und ermitteln Sie A und B.

Beispiel A.1.1 Die Fehlerfolge ε_n bei der Berechnung einer Nullstelle x^* einer Funktion $f \in \mathbb{C}^2([a,b])$ mit Hilfe des Sekantenverfahrens kann näherungsweise durch die Rekursionsformel

$$\varepsilon_{n+1} = u \cdot \varepsilon_n \cdot \varepsilon_{n-1}$$

mit $u > 0$ dargestellt werden. Setzt man nun $a_k = \log(\varepsilon_k)$, so ergibt sich die inhomogene lineare Differenzengleichung

$$a_{n+1} = \log(u) + a_n + a_{n-1} . \tag{A.6}$$

Wie bei der Berechnung der Fibonacci-Zahlen sind

$$a_n^{(h)} = c_1 \left(\frac{1-\sqrt{5}}{2} \right)^n + c_2 \left(\frac{1+\sqrt{5}}{2} \right)^n$$

die Lösungen der homogenen Differenzengleichung und

$$a_n^{(p)} = -\log(u)$$

eine partikuläre Lösung von (A.6). Das führt zu

$$a_n = a_n^{(p)} + a_n^{(h)} = -\log(u) + c_1 \left(\frac{1-\sqrt{5}}{2} \right)^n + c_2 \left(\frac{1+\sqrt{5}}{2} \right)^n$$

und man erhält schließlich

$$\varepsilon_n = \frac{1}{u} \cdot e^{c_1 (\frac{1-\sqrt{5}}{2})^n + c_2 (\frac{1+\sqrt{5}}{2})^n} .$$

Mit $\lambda_1 := \frac{1-\sqrt{5}}{2}$ und $\lambda_2 := \frac{1+\sqrt{5}}{2}$ ist

$$\frac{\varepsilon_{n+1}}{\varepsilon_n^{\lambda_2}} = e^{c_1\lambda_1^{n+1}+c_2\lambda_2^{n+1}-c_1\lambda_1^n\lambda_2-c_2\lambda_2^{n+1}} = e^{-\sqrt{5}c_1\lambda_1^n} \longrightarrow 1 \quad (n \to \infty)$$

in Abhängigkeit von n beschränkt. Also ist bei Konvergenz gegen 0 die maximale Konvergenzordnung gleich $\frac{1+\sqrt{5}}{2}$.

Literaturverzeichnis

[AS] M. ABRAMOWITZ, I.A. STEGUN Pocketbook of Mathematical Functions with Formulas, Verlag Harri Deutsch, Frankfurt/Main (1984).

[A] R.A. ADAMS, “Sobolev Spaces”, Pure Appl. Math. 65, Academic Press, New York, 1975.

[Calvetti] D. CALVETTI, G.H. GOLUB, W.B. GRAGG UND L. REICHEL, Computation of Gauss-Kronrod rules. Math. Comp. 69, 1035–1052 (2000).

[Cull] P. CULL, M. FLAHIVE UND R. ROBSON, Difference equations, Undergraduate Texts in Mathematics. Springer, New York (2005).

[Cuyt/Wuytack] A. CUYT, L. WUYTACK, Nonlinear Methods in Numerical Analysis, North-Holland (Amsterdam).

[Elayadi] S. ELAYADI An introduction to difference equations, Undergraduate Texts in Mathematics. Springer, New York (2005).

[Golub] G. H. GOLUB, C. F. VAN LOAN, Matrix Computations, 3. ed., Hopkins Univ. Press, 1996.

[H] W. HACKBUSCH, Iterative Lösung großer schwachbesetzter Gleichungssysteme, 2. Auflage, Teubner-Verlag, Stuttgart, 1993.

[Hämmerlin/Hoffmann] G. HÄMMERLIN, K.-H. HOFFMANN, Numerische Mathematik, 4. Auflage, Springer-Verlag, Berlin u.a., 1994.

[Heuser] H. Heuser: Funktionalanalysis. Teubner, 3. Auflage, 1992.

[Kiefer] J. KIEFER, Optimum sequential search and approximation methods under minimum regularity assumptions. J. Soc. Ind. Appl. Math. 5, 105-136 (1957).

[Kronrod] A.S. KRONROD, Nodes and weights of quadrature formulas. Sixteen-place tables. New York: Consultants Bureau. Authorized translation from the Russian (1965).

[Laurie] D.P. LAURIE, Calculation of Gauss-Kronrod quadratur rules. Math. Comp. 1133-1145 (66) 1997.

[Lebed] G. K. LEBED, Quadrature formulas with minimum error for certain classes of functions, Mathematical Notes 3, 368-373 (1968).

[Marsden] M. J. MARSDEN, An identity for spline functions with applications to variation-dimishing spline approximation. J. Approx. Theory 3 (1970), 7-49.

[Meyberg/Vachenauer] K. MEYBERG, P. VACHENAUER, Höhere Mathematik 1. Springer, Berlin (1999)

[Plato] R. PLATO, Numerische Mathematik kompakt, Vieweg-Verlag.

[QSS1] A. QUARTERONI, R. SACCO, F. SALERI, Numerische Mathematik 1, Springer 2002.

[QSS2] A. QUARTERONI, R. SACCO, F. SALERI, Numerische Mathematik 2, Springer 2002.

[Rivlin] T.J. RIVLIN, An Introduction to the Approximation of Functions, Blaisdell Publ., Waltham, MA, 1969.

[Schönhage] A. SCHÖNHAGE, Approximationstheorie, de Gruyter, Berlin, 1971.

[Schwarz] H. R. SCHWARZ, Numerische Mathematik, 4. Auflage, Teubner-Verlag, Stuttgart, 1997.

[Stör/Bulirsch] J. STÖR, R. BULIRSCH, Numerische Mathematik 1 und 2, Springer-Verlag, Berlin u.a., 1994.

[SW] K. STREHMEL, R. WEINER, Numerik gewöhnlicher Differentialgleichungen, Teubner-Verlag, Stuttgart, 1995.

[St] A. H STROUD, Gaussian quadrature formulas, Prentice-Hall, Englewood Cliff (1966).

[Sz] G. SZEGÖ, Orthogonal Polynomials, AMS 3. Auflage, 1967.

[TS] W. TÖRNIG, P. SPELLUCCI, Numerische Mathematik für Ingenieure und Physiker, Band 1 & 2, Springer-Verlag, Berlin u.a.

[W] W. WALTER, Gewöhnliche Differentialgleichungen, 6. Auflage, Springer-Verlag, Berlin u.a., 1996.