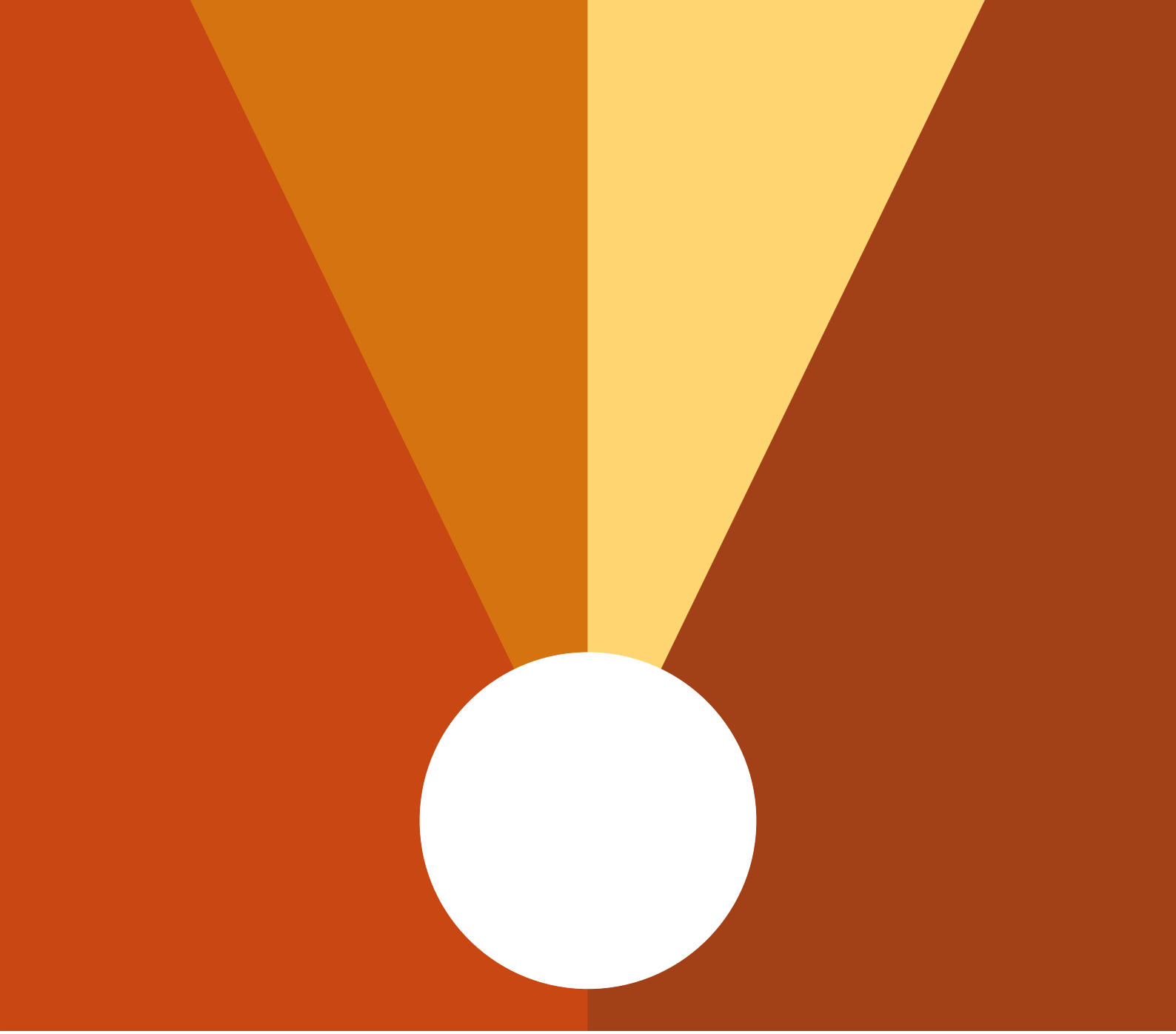


Numerische Analysis

Stefan Funken
Universität Ulm

Sommersemester 2021

Version 3. Juli 2021

Präambel

Die Numerische Analysis ist ein Teil der numerischen Mathematik, der verwendet wird, um Näherungen für schwierige Probleme zu finden, wie z. B. das Finden der Nullstellen nichtlinearer Gleichungen oder die Integration komplizierter Funktionen, für die es keine analytischen Lösungen gibt. Sie wird in einer Vielzahl von Disziplinen angewandt, z. B. in den Wirtschaftswissenschaften, in allen Bereichen der Ingenieurwissenschaften, in der Informatik und in anderen Bereichen.

Dieses Skript ist eine Neuüberarbeitung der Vorlesunginhalte, angepasst an ein modernes online-Studium. Es soll die einzelnen Themen motivieren, die wichtigsten Ideen des Fachgebiets vorstellen, eine mathematische Analyse der Methoden verständlich präsentieren und einen modernen Umgang mit Algorithmen durch zahlreiche numerische Experimente bieten.

Kapitel, die mit einem Stern markiert sind, beinhalten ergänzendes Material. Notwendig zum Verstehen der Vorlesung sind Kenntnisse aus dem Modul „Analysis“ und Programmierkenntnisse in MATLAB.

MATLAB

In der Vorlesung und diesem Skript verwenden wir MATLAB zur Darstellung von Algorithmen. Es ist nicht unbedingt erforderlich, dass hier eine bestimmte Programmiersprache verwendet wird, aber die wachsende Präsenz von MATLAB in den verschiedensten Wissenschaftsabteilungen zeigt, dass eine gemeinsame Sprache viele Hindernisse glätten kann. Mit MATLAB werden alle Probleme zwischen Datenein- und -ausgabe, Plotten usw. auf einen Schlag gelöst. Datenstrukturen werden standardisiert, indem man sich auf einheitliche Befehle stützt. Diese Ziele können alle auf andere Weise erreicht werden. Es ist jedoch hilfreich, ein Paket zu haben, das auf fast allen Betriebssystemen läuft und die Details vereinfacht, so dass sich die Studenten auf die wirklichen mathematischen Probleme konzentrieren können.

Anhang B ist ein MATLAB-Tutorial, das als erste Einführung für Studierende oder als Referenz für bereits Vertraute verwendet werden kann.

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.

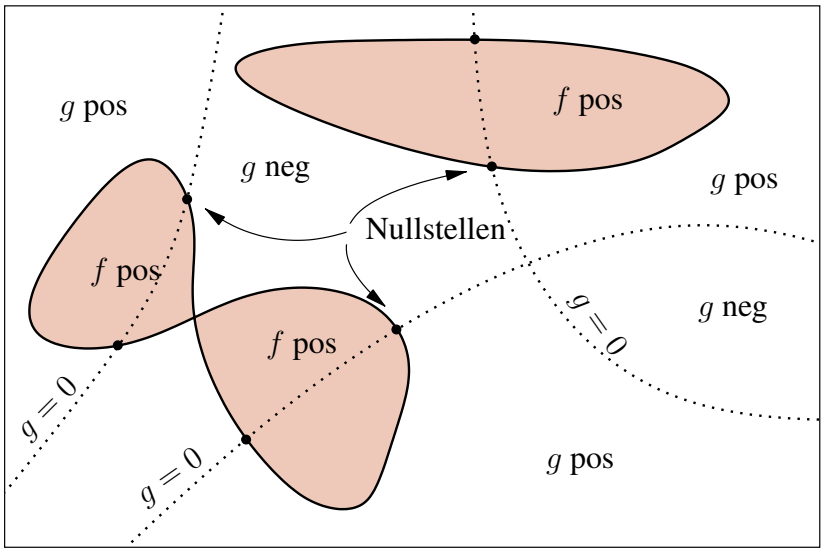


Abbildung 1.1.: Lösung von zwei Gleichungen in zwei Unbekannten. Die durchgezogenen Linien sind Niveaulinien zu $f(x, y) = 0$, die gestrichelten Linien zu $g(x, y) = 0$. Die gesuchten Lösungen sind die Schnittpunkte der völlig unabhängigen Nulllinien. Die Anzahl der Nullstellen ist im Allgemeinen a-priori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

Ist 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. Mehrere Modellbeispiele, bei denen nichtlineare Gleichungen auftreten, wollen wir hier vorstellen.

■ **Beispiel 1.1 — 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 400 Euro monatlich aufbringen kann? Oder allgemeiner eine Kreditsumme K_0 in n Jahren mit monatlichen Raten R getilgt habe 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?

■

■ **Beispiel 1.2 — Nullstellen von Orthogonalpolynomen.** Die **Legendre-Polynome** $P_n \in \mathbb{P}_n$ $(n = 0, 1, 2, \ldots)$ erfüllen die Drei-Term-Rekursion

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

Die ersten Legendre-Polynome lauten

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

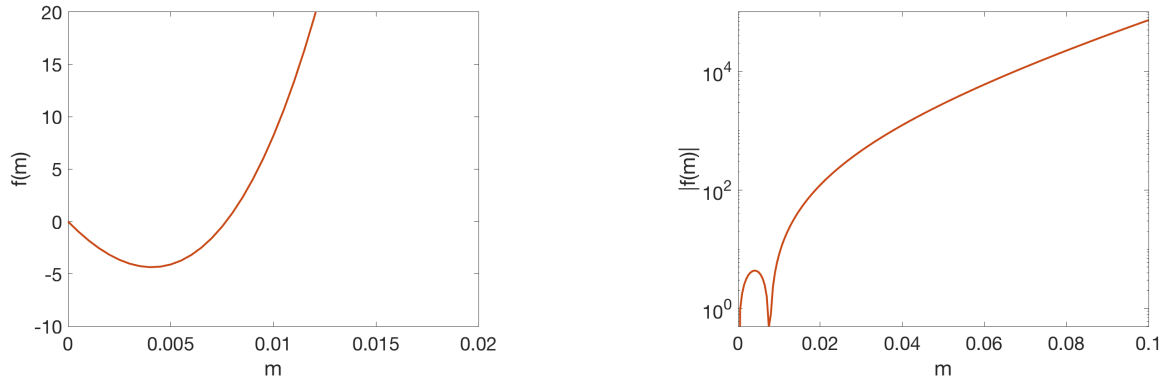


Abbildung 1.2.: Der Graph von $f(m) := (m\,K_0 - R)(1+m)^n + R$ (links, y-Achse linear skaliert, $x \in [0, 0.02]$) bzw. $|f(m)|$ (rechts, logarithmisch-skaliert, $x \in [0, 0.1]$) für $K_0 = 10000$, $R = 250$ und $n = 48$. Die Funktion f hat eine Nullstelle bei 0 und bei ≈ 0.008.

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. P_1 hat 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 Theorem **??** in Kapitel 5). Die Berechnung der Nullstellen ist u.a. wichtig im Zusammenhang mit Gauß-Quadraturformeln. ∎

∎ **Beispiel 1.3 — Extremalstellen von skalaren Funktionen.** Die mehrdimensionale Erweiterung der Rosenbrock[1]-Funktion

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

hat für $n \geq 4$ mindestens ein lokales Minimum in der Umgebung von $(x_1, x_2, \ldots, x_n) = (-1, 1, \ldots, 1)$ neben dem globalen Minimum $(x_1, \ldots, x_n) = (1, \ldots, 1)$. 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 $x \in \mathbb{R}^n \setminus \{(1, \ldots, 1)\}$ ist zu bestimmen mit

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

∎

Betrachten wir dazu zunächst die Situation in einer Raumdimension.

[1] Howard Harry Rosenbrock, 16. Dez. 1920 - 21 Okt. 2010

1.1. Bisektionsmethode

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 $f(a) \cdot f(b) < 0$.

Aus Stetigkeitsgründen existiert eine Lösung x^* im Inneren von I mit $f(x^*) = 0$.
Durch den Mittelpunkt $m = \frac{1}{2}(a + b)$ des Intervalls I wird 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. Wir erhalten damit folgenden Algorithmus:

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;
 end
 temp = [temp;m];
end
% Loesung
Nullstelle = m;
```

MATLAB-Beispiel:

Testen wir nun die Bisektionsmethode anhand von Bsp. 1.1. Wie hoch darf der Zinssatz sein, wenn man einen Kredit über 10.000$ in 48 Monaten zurückzahlen möchte, aber nur 250$ 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.

Definition 1.4 — Konvergenzgeschwindigkeit. Sei (x_k) eine reellwertige konvergente Folge mit $x_k \in \mathbb{R}^n$ $(k \in \mathbb{N})$ und Grenzwert $x \in \mathbb{R}^n$. Man bezeichnet (x_k) als **linear konvergent**, wenn es ein $0 < \varrho < 1$ gibt mit

$$\|x - x_{k+1}\| \leq \varrho \|x - x_k\| \quad (k = 0, 1, \ldots).$$

Die Zahl ϱ wird **Konvergenzfaktor** (oder auch **Kontraktionsrate**) genannt. Gibt es eine gegen Null konvergente Folge (ϱ_k) mit

$$\|x - x_{k+1}\| \leq \varrho_k \|x - x_k\| \quad (k = 0, 1, \ldots),$$

so heißt (x_k) **superlinear konvergent**. Gibt es ein $\varrho > 0$ und ein $1 < q \in \mathbb{R}$ mit

$$\|x - x_{k+1}\| \leq \varrho \|x - x_k\|^q \quad (k = 0, 1, \ldots),$$

so heißt (x_k) konvergent mit **Konvergenzordnung** q. Für $q = 2$ spricht man auch von **quadratischer Konvergenz**.

Bemerkung 1.5 — 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

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

1.2. 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

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

durch $(a, f(a))$ und $(b, f(b))$. Mit den beiden Intervallen $[a, \xi]$, $[\xi, b]$ verfahren wir nun analog zur Methode der Intervallhalbierung, wobei gilt:

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

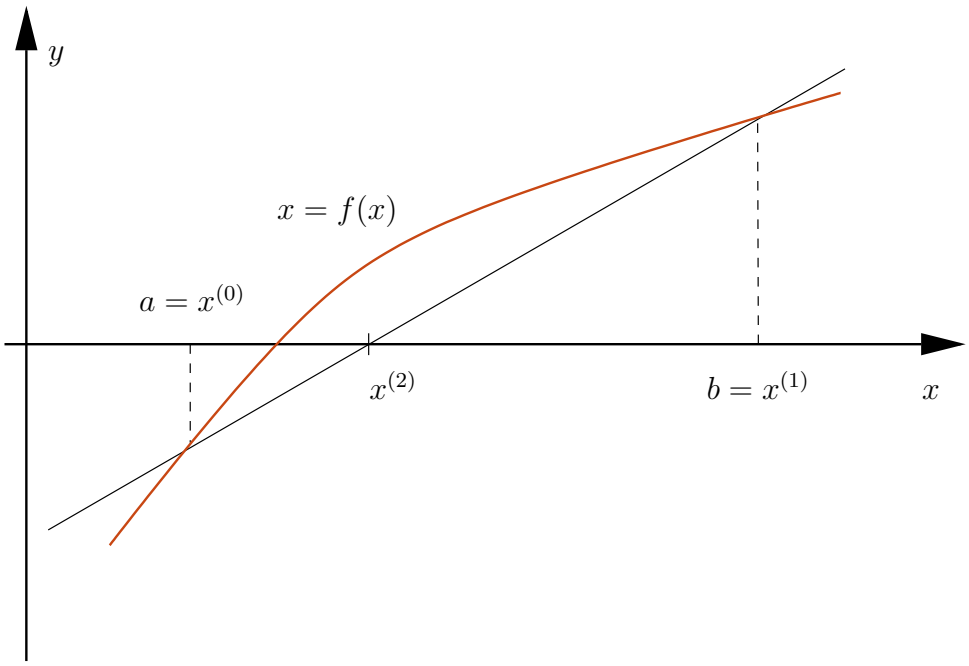


Abbildung 1.3.: Die geometrische Interpretation der Regula-Falsi-Methode.

Als Algorithmus ergibt sich somit:

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; % Loesung
```

Satz 1.6 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) die sich aus aus obigem Algorithmus **??** 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) linear. Analoges gilt für konvexe Funktionen mit ε_k^b anstatt ε_k^a. ■

MATLAB-Beispiel:

[2] Taylor, Brook (1685-1731)

Testen wir nun das Regula-Falsi-Verfahren anhand Bsp. 1.1 $K_0 = 10000$, $n = 48$ 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.

1.3. Die Sekantenmethode

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

Zu zwei vorgegebenen Näherungswerten x_0 und x_1, welche die Lösung nicht notwendigerweise einzuschließen brauchen, bestimmt man x_2 als Nullstelle der Sekante 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

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

Dieses zweistufige Iterationsverfahren setzt natürlich $f(x_k) \neq f(x_{k-1})$ voraus und gehört nicht zur Klasse der oben betrachteten Iterationsverfahren.

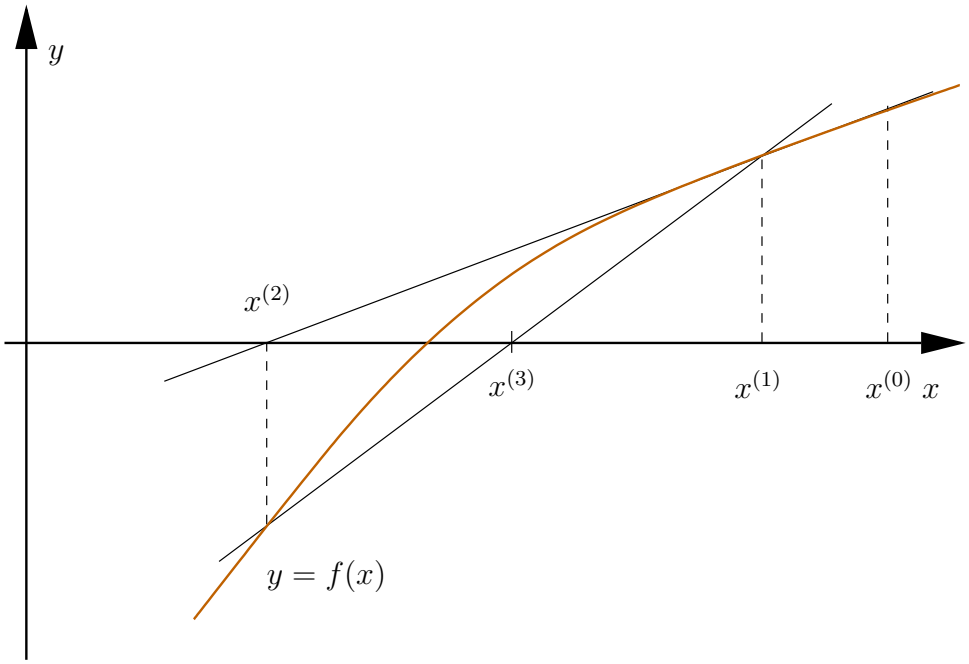


Abbildung 1.4.: Die geometrische Interpretation der Sekantenmethode.

Satz 1.7 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

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

zu lösen. Einsetzen ergibt

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

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

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

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 $\varrho = C^{1/p} = C^{0.618}$. ∎

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 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.

1.4. Das Verfahren von *Newton*

Herleitung des Algorithmus Ist die gegebene Funktion $f(x)$ der zu lösenden Gleichung $f(x) = 0$ stetig differenzierbar, so wird im Verfahren von *Newton*[3] 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).

[3] Newton, Isaac (1642-1727)

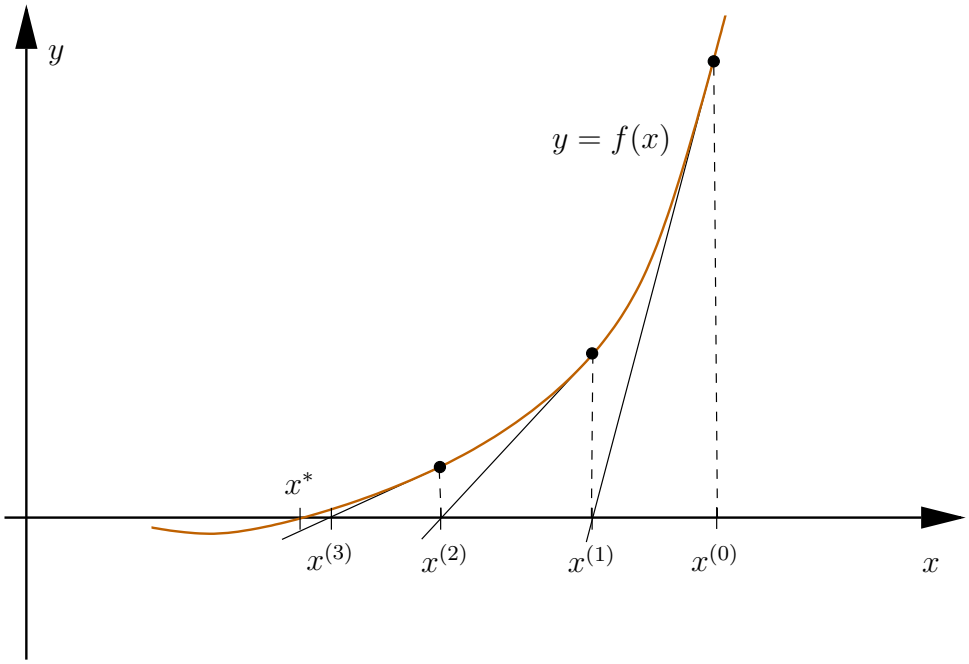


Abbildung 1.5.: Die geometrische Interpretation des *Newton*-Verfahrens.

Aus der Tangentengleichung

$$y(x) = f(x_k) + (x - x_k)f'(x_k)$$

ergibt sich die Iterationsvorschrift

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}\,. \tag{1.6}$$

Bemerkung 1.8 Die Methode von *Newton* gehört zur Klasse der Fixpunktiterationen mit der Funktion

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

Satz 1.9 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 $\phi : I_\delta \to I_\delta$ eine Kontraktion darstellt. Ferner ist für jeden Startwert $x_0 \in I_\delta$ die Konvergenz der Folge (x_k) 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

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

gilt. Somit ist ϕ eine Kontraktion in I_δ. Weiterhin sind für I_δ die Voraussetzungen des *Banachschen* Fixpunktsatzes erfüllt und damit ist die Konvergenz von (x_k) gezeigt.
Zum Beweis der Konvergenzordnung definieren wir $e_{k+1} := x_{k+1} - x^*$. Eine *Taylorentwicklung* 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. ■

Betrachten wir nun den allgemeinen Fall von Systemen nichtlinearer Gleichungen ($n \geq 2$). Die *Jacobi*-Matrix oder Funktionalmatrix der Abbildung $f = (f_1, \ldots, f_n) : \mathbb{R}^n \to \mathbb{R}^n$ ist gegeben durch

$$J_f(x) := \begin{pmatrix} \nabla f_1 \\ \vdots \\ \nabla f_n \end{pmatrix} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \end{pmatrix}.$$

Die *Taylor*-Entwicklung von f um einen Startwert $x^{(0)}$ ergibt in diesem Fall

$$0 = f(x^*) = \underbrace{f(x^{(0)}) + J_f(x^{(0)})(x^* - x^{(0)})}_{=:\overline{f}(x)} + o(\|x^* - x^{(0)}\|) \quad \text{für } x^* \to x^{(0)} \,.$$

Die Nullstelle $x^{(1)}$ der linearisierten Abbildung $\overline{f}(x)$ ist jedoch gerade

$$x^{(1)} = x^{(0)} - J_f(x^{(0)})^{-1} f(x^{(0)}),$$

falls $det(J_f(x^{(0)})) \neq 0$ erfüllt ist.

Dies motiviert die *Newton*-Iteration

$$J_f(x^{(k)})s^{(k)} = -f(x^{(k)}) \tag{1.7}$$
$$x^{(k+1)} = x^{(k)} + s^{(k)}. \tag{1.8}$$

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
```

Bemerkung 1.10 Wir haben die numerische Lösung eines nichtlinearen Gleichungssystems auf die numerische Lösung einer Folge von linearen Gleichungssystemen übertragen.

■ **Beispiel 1.11** Vergleichen wir nun die Bisektionsmethode, das Sekantenverfahren und das Newton-Verfahren angewandt auf das Problem aus Bsp. 1.1 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. ?? dargestellte Konvergenz. ■

Bemerkung 1.12 — 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.

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

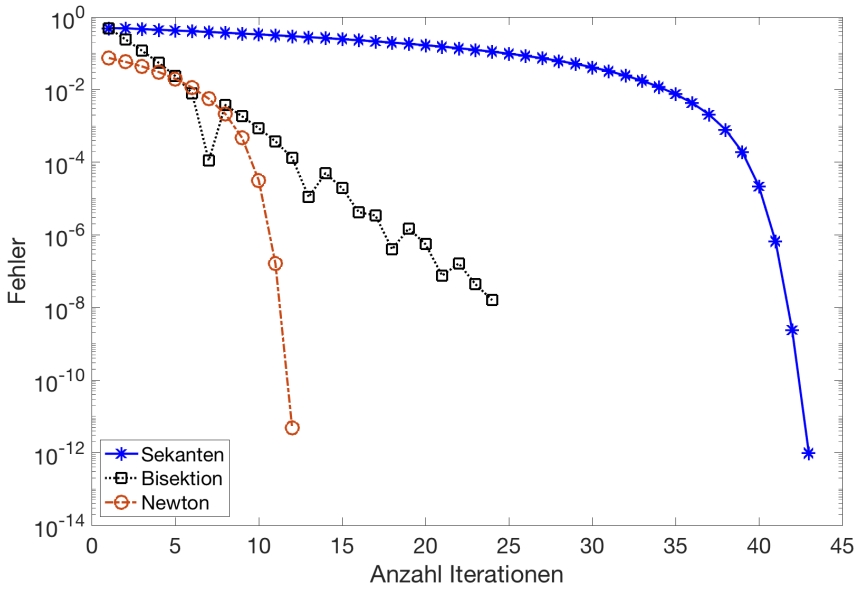


Abbildung 1.6.: Konvergenz von Bisektions-, Sekanten- und Newton-Verfahren angewandt auf $f(m) = (m \cdot K_0 - R) \cdot (1+m)^n + R$ aus Bsp. 1.1 mit $K_0 = 10000$, $n = 48$ und $R = 250$.

Satz 1.13 Sei $D \subset \mathbb{R}^n$ offen und konvex, $f : D \to \mathbb{R}^n$ eine stetig partiell differenzierbare Funktion mit invertierbarer *Jacobi*-Matrix $J(x)$ für alle $x \in D$. Es gelte ferner für ein $\omega \geq 0$ die folgende *Lipschitz*-Bedingung:

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

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

$$\varrho := \|x^* - x^{(0)}\| < \frac{2}{\omega} \quad \text{und } \mathcal{U}_\varrho(x^*) \subseteq D.$$

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

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

und

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

Die Konvergenzgeschwindigkeit läßt sich abschätzen durch

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

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

Beweis. Als ersten Schritt des Beweises leiten wir aus der *Lipschitz*-Bedingung für $x, y \in D$

her, dass gilt:

$$\|J^{-1}(x)(f(y)-f(x)-J(x)(y-x))\| \le \frac{\omega}{2}\|y-x\|^2. \tag{1.9}$$

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

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

Da auch $J^{-1}(x)$ unabhängig von s ist, erhalten wir für die linke Seite von (1.9)

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

Nun erhalten wir für die Iterationsvorschrift

$$x^{(k+1)} = x^{(k)} - J^{-1}(x^{(k)})f(x^{(k)})$$

sowie $f(x^*)=0$

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

Mit (1.9) erhalten wir nun

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

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

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

und somit induktiv für $k>0$

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

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

Zum Beweis der Eindeutigkeit in der Umgebung $\mathcal{U}_{2/\omega}(x^*)$ um x^* mit Radius $2/\omega$ benutzen wir nochmals (1.9).

Sei $x^{**} \in \mathcal{U}_{2/\omega}(x^*)$ eine weitere Lösung, also $f(x^{**}) = 0$ und $\|x^* - x^{**}\| < 2/\omega$. Einsetzen in (1.9) liefert

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

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

Bemerkung 1.14 — Merkregel. Das *Newton*-Verfahren konvergiert lokal quadratisch.

Bemerkung 1.15 — Konvergenztest. Als Näherung an den Fehler $\|x - x^{(k)}\|$ verwenden wir den Term $\|J^{-1}(x^{(k)})f(x^{(k)})\|$ $(= \|J^{-1}(x^{(k)})(f(x) - f(x^{(k)}))\|)$. Da man erwartet, dass der Fehler monoton fällt, d.h. $\|x - x^{(k+1)}\| \leq \|x - x^{(k)}\|$ gilt, testet man dieses für jedes k durch den **natürlichen Monotonietest**, d.h.

$$\|J^{-1}(x^{(k)})f(x^{(k+1)})\| \leq \overline{\theta}\,\|J^{-1}(x^{(k)})f(x^{(k)})\| \quad \text{sei erfüllt für ein } \overline{\theta} < 1. \tag{1.10}$$

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

$$J(x^{(k)})s^{(k)} = -f(x^{(k)}) \quad \text{und } x^{(k+1)} = x^{(k)} + s^{(k)}.$$

Somit ist der Ausdruck $J^{-1}(x^{(k)})f(x^{(k)})$ auf der rechten Seite in (1.10) gleich dem Negativen der Newton-Korrektur $s^{(k)}$, die sowieso berechnet werden muss. Zusätzlich muss nur $\overline{s}^{(k)}$, definiert durch

$$J(x^{(k)})\overline{s}^{(k)} = f(x^{(k+1)})\,,$$

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.

Bemerkung 1.16 — Dämpfung des Newtonverfahrens. 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.

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

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 $\lambda_k/2$.

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
```

■ **Beispiel 1.17 — 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 1.3, 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 $(x_1, \ldots, x_n) = (1, \ldots, 1)$ mit dem Newton-Verfahren ein weiteres lokales Minimum in der Umgebung von $(x_1, x_2, \ldots, x_n) = (-1, 1, \ldots, 1)$ von f in Bsp. 1.3 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 1.18 — 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 $u \in W_0^{1,p}(\Omega) := \{u \in L^p | \nabla u \in (L^p(\Omega))^d, u|_{\partial\Omega} = 0\}$ mit

$$\begin{aligned} \operatorname{div}(|\nabla u|^{p-2}\nabla u) &= f \quad \text{im Gebiet } \Omega, \\ u &= 0 \quad \text{auf dem Rand } \Gamma := \partial\Omega, \end{aligned} \tag{1.11}$$

wobei div den Divergenzoperator [4]. bezeichne.

Die Lösung des p-Laplace-Problems (1.11) ist der Minimierer des Energiefunktionals

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

über alle Funktionen aus dem Sobolev[5]-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

$$\varphi_i(x) := \begin{cases} \frac{x - x_{i-1}}{x_i - x_{i-1}} & , \text{falls } x \in [x_{i-1}, x_i], \\ \frac{x_{i+1} - x}{x_{i+1} - x_i} & , \text{falls } x \in (x_i, x_{i+1}], \\ 0 & , \text{sonst} \end{cases} \tag{1.12}$$

[4] Für $u = (u_1, \ldots, u_n) \in C^1(\Omega; R^n)$ sei $\operatorname{div}(u) := \sum_{i=1}^n \frac{\partial u_i}{\partial x_i}$

[5] Sergei Lvovich Sobolev, 1908 - 1989

definiert sogenante 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

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

minimiert, mit $\alpha = (\alpha_1, \dots \alpha_n)$. Man beachte $u_N'(x)|_{(x_{k-1}, x_k)} = h_k^{-1}(\alpha_{k-1} - \alpha_k)$ und

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

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

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

erfüllt. Dies führt zu dem nichtlinearen Gleichungssystem

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

mit

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

Für $f \equiv 1$ erhalten wir

$$u(x) = \frac{p-1}{p} \left(1 - x^{\frac{p}{p-1}}\right) \quad 1 < p < \infty \tag{1.13}$$

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

$$u(x) - u_N(x) = \frac{\frac{p-1}{p} - 2^{\frac{1}{1-p}}}{M^2} \left(1 - |x|^{(2-p)/(p-1)}\right) \tag{1.14}$$

zeigen.

Aufgabe 1.19 Man zeige (1.13) und (1.14). ■

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

MATLAB-Funktion: 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.

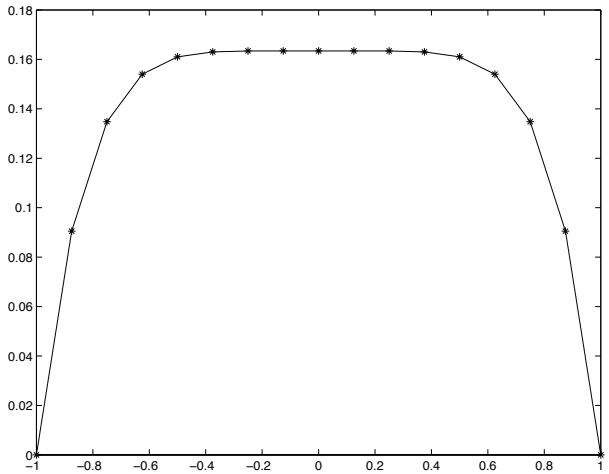


Abbildung 1.7.: Näherungsweise Lösung des p-Laplace-Problems mit $p = 1.2$ und $N = 15$.

■

1.5. 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*[6] 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

[6] Broyden, Charles George (1933 -)

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

$$B_k s^{(k)} = -f(x^{(k)}) \tag{1.15}$$

berechnet, wobei $x^{(k+1)} = x^{(k)} + s^{(k)}$ sei. Im Eindimensionalen liefert das Sekantenverfahren zu zwei gegebenen Werten $x^{(-1)}$ und $x^{(0)}$ iterativ die Steigung der Sekante durch

$$\beta_k = \frac{f(x^{(k)}) - f(x^{(k-1)})}{x^{(k)} - x^{(k-1)}} \quad (k \geq 0) \tag{1.16}$$

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 (1.16) an B_k lautet

$$B_k s^{(k-1)} = f(x^{(k)}) - f(x^{(k-1)}) =: \delta f_k\,.$$

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

$$B_k(x^{(k)} - x^{(k-j)}) = f(x^{(k)}) - f(x^{(k-j)}) \quad (j = 1, \ldots, n) \tag{1.17}$$

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

$$d_k := f(x^{(k)}) + B_k(x - x^{(k)}) - \left(f(x^{(k-1)}) + B_{k-1}(x - x^{(k-1)})\right)\,, \tag{1.18}$$

in der euklidischen Norm minimiert wird. Setzen wir $j = 1$ in (1.17) so wird (1.18) zu

$$d_k = (B_k - B_{k-1})(x - x^{(k-1)}). \tag{1.19}$$

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 (1.19)

$$d_k = \alpha(B_k - B_{k-1})s^{(k-1)} + (B_k - B_{k-1})r\,. \tag{1.20}$$

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 (1.20) unabhängig von B_k und es bleibt nur der zweite Term in (1.20) 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 (1.17) gilt, kann rekursiv mittels des Rang-1-Updates von B_{k-1}

$$B_k = B_{k-1} + \frac{\left(\delta f_k - B_{k-1}s^{(k-1)}\right)\left(s^{(k-1)}\right)^T}{\left(s^{(k-1)}\right)^T s^{(k-1)}} \tag{1.21}$$

berechnet werden. Die Methode (1.15) mit der Wahl (1.21) 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 1.20 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. 1.3 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 **??** zeigt das Verhalten der Euklidischen Norm des Fehlers beider Methoden.

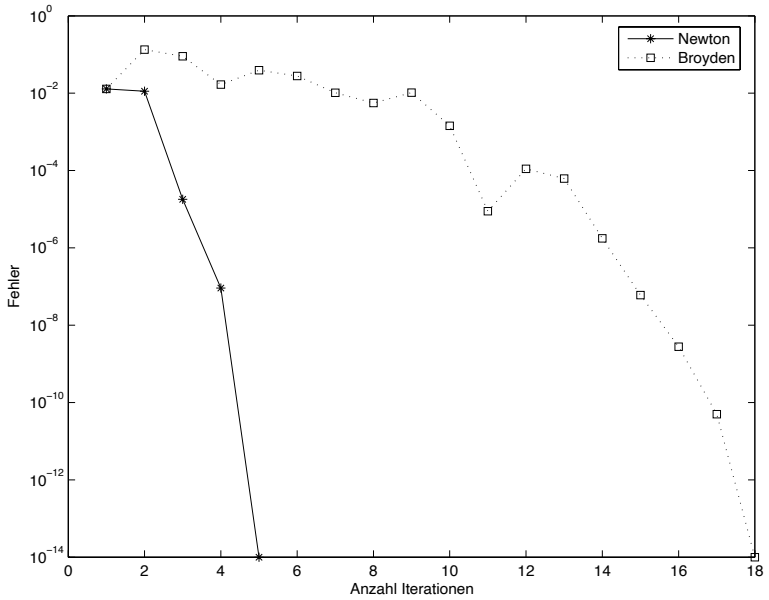


Abbildung 1.8.: Euklidische Norm des Fehlers für das Broyden- und Newton-Verfahren im Fall des nichtlinearen Problems aus Bsp. 1.3 für $n = 6$.

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 2.1 (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 je nach Anwendung **Polynome**, **trigonometrische Funktionen**, **Exponentialfunktionen** sowie **rationale Funktionen**.

2.1. Klassische Polynom-Interpolation

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

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

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

welches die Interpolationsbedingungen

$$P(x_i) = f_i \quad (i = 0, \ldots, n) \tag{2.1}$$

erfüllt.

Die Frage nach der Existenz eines solchen Polynoms P führt uns zu folgendem wichtigen Resultat:

Satz 2.2 — Existenz und Eindeutigkeit der Polynominterpolation. Zu beliebigen $(n+1)$ Stützstellen (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 (2.1) erfüllt und höchstens den Grad n besitzt.

Beweis. **(Existenz:)** Die Existenz des Interpolationspolynoms P zeigen wir auf konstruktive Art. Zu diesem Zweck betrachten wir zu den gegebenen Stützstellen die $(n+1)$ **Lagrange**-Polynome

$$L_i(x) := \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{(x - x_j)}{(x_i - x_j)}. \tag{2.2}$$

Diese Polynome sind vom echten Grad n und besitzen offensichtlich die Eigenschaft

$$L_i(x_k) = \delta_{ik} = \begin{cases} 1 & \text{für } i = k, \\ 0 & \text{für } i \neq k. \end{cases} \tag{2.3}$$

Demzufolge besitzt das Polynom

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

die geforderten Interpolationseigenschaften. Wegen (2.3) gilt:

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

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 $P(x)$ und $Q(x)$ zwei Polynome jeweils vom Grad höchstens gleich n mit

$$P(x_k) = Q(x_k) = f_k \quad (k = 0, 1, \ldots, n). \tag{2.4}$$

Aus dieser Eigenschaft (2.4) folgt, dass $D(x) := P(x) - Q(x)$ ein Polynom vom Grad kleiner oder gleich n ist 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 2.3 Die **Lagrange**-Darstellung (2.2) ist für praktische Zwecke meist zu rechenaufwendig, sie eignet sich jedoch hervorragend für theoretische Fragestellungen.

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

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

Bemerkung 2.4 — Vandermonde-Matrix. Die Interpolationsbedingungen $P(x_i) = f_i$ lassen sich nun als folgendes lineares Gleichungssystem auffassen:

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

In Numerik I 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 $5n^2/2 + \mathcal{O}(n)$).

Bemerkung 2.5 Die Darstellung in monomialer Basis ist numerisch instabil und sollte daher für große n im Allgemeinen nicht verwendet werden.

2.2. *Hermite*-Interpolation und dividierte Differenzen

Definition 2.6 Das nach Satz 2.2 eindeutig bestimmte Polynom P heißt **Interpolationspolynom** von f zu den paarweise verschiedenen Stützstellen $x_0, \dots, x_n$ und wird mit

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

bezeichnet.

Bemerkung 2.7 Ist man nur an der Auswertung des Interpolationspolynoms P an einer Stelle x interessiert, so muss man dazu nicht erst P bestimmen, sondern kann $P(x)$ durch rekursive Berechnung effektiver (d.h vor allem effektiver bezüglich des Aufwands) bestimmen. Motiviert wir dies im Folgenden durch das Lemma von *Aitken*.

Lemma 2.8 — von Aitken. Für das Interpolationspolynom $P = P(f|x_0, \dots, x_n)$ gilt die Rekursionsformel

$$P(f|x_0, \dots, x_n)(x) = \frac{(x_0 - x)P(f|x_1, \dots, x_n)(x) - (x_n - x)P(f|x_0, \dots, x_{n-1})(x)}{x_0 - x_n}. \quad (2.5)$$

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

Beweis. Sei $\phi(x)$ definiert als der Term auf der rechten Seite von (2.5). Dann ist $\phi \in \mathbb{P}_n$ und es gilt:

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

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

Herleitung des Algorithmus von *Aitken* und *Neville*[1]

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

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

Die Rekursion (2.5) schreibt sich nun

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

oder allgemeiner für P_{ik} $(i \geq k)$:

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

[1] Neville, Alexander Craig (1895-1967)

Nach dem Schema von *Neville* lässt sich P_{nn} ausgehend von den Daten $(f_0 \dots, f_n)$ wie folgt berechnen:

$$
\begin{array}{lllllllllll}
f_0 & = & P_{00} & & & & & & & & \\
 & & & \searrow & & & & & & & \\
f_1 & = & P_{10} & \to & P_{11} & & & & & & \\
\vdots & & \vdots & & \vdots & & & & & & \\
\vdots & & \vdots & & & & & & & & \\
 & & & & & \searrow & & & & & \\
\vdots & & \vdots & & \vdots & \to & P_{n-2,n-2} & & & & \\
 & & & & & \searrow & & \searrow & & & \\
f_{n-1} & = & P_{n-1,0} & \to & \dots & \to & P_{n-1,n-2} & \to & P_{n-1,n-1} & & \\
 & & & \searrow & & \searrow & & \searrow & & \searrow & \\
f_n & = & P_{n,0} & \to & P_{n,1} \dots & \to & P_{n,n-2} & \to & P_{n,n-1} & \to & P_{nn}
\end{array}
$$

Daraus gewinnen wir die folgende Rechenvorschrift:

Algorithmus 2.2.1: *Aitken-Neville*

$P(j,0) = f(j)$

$P(j,k) = P(j,k-1) + \frac{x-x(j)}{x(j)-x(j-k)}(P(j,k-1) - P(j-1,k-1)), \quad j \geq k$

---

oder als Matlab-Routine (man beachte die Indexverschiebung) um $P(f|x_0, \dots, x_n)(x)$ an einer Stelle x auszuwerten.

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(0,1,3);
>> AitkenNeville(x,x.^2,2)
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. 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 2.9 Die rekursive Struktur des Algorithmus lässt sich auch dazu nutzen, das gesamte Polynom $P(f|x_0, \ldots, x_n)$ zu bestimmen.

Dies gilt auch für die verallgemeinerte Interpolationsaufgabe, bei der neben den Funktionswerten $f(x_i)$ auch die Ableitungen gegeben sind, der sogenannten **Hermite**-Interpolation. Um diese einzuführen, benötigen wir noch einige Notationen.

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

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

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

Gleiche Knoten nummerieren wir hierbei mit

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

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

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

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

$$\mu_i(P) = \mu_i(f) \quad (i = 0, \ldots, n). \tag{2.6}$$

Die Lösung $P = P(f|x_0, \ldots, x_n) \in \mathbb{P}_n$ von (2.6) heißt **Hermite**-Interpolierende.

Satz 2.10 — Existenz und Eindeutigkeit. 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:

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

Beweis. Die Abbildung

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

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. ■

Definition 2.11 — Newton-Basis. Es seien $x_0, \ldots x_{n-1} \in \mathbb{R}$ 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 $\omega_0, \ldots, \omega_n$ als **Newton**-Basis des Polynomraums $\mathbb{P}_n$ mit Basiselementen ω_i.

Bemerkung 2.12 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 die Hermite-Interpolationsaufgabe effizient lösen lässt, verwenden wir die Darstellung des Interpolati-

onspolynoms in der Newton-Basis, d.h.

$$\begin{aligned} P(x) &= a_0 + a_1(x-x_0) + a_2(x-x_0)(x-x_1) + \ldots + a_n \prod_{j=0}^{n-1}(x-x_j) \\ &= \sum_{i=0}^{n} a_i \omega_i(x), \end{aligned}$$

wobei sich die unbekannten Koeffizienten $a_0, \ldots, a_n$ prinzipiell aus den Interpolationsbedingungen

$$\begin{array}{rcll} P(x_0) &=& a_0 & = f(x_0) \\ P(x_1) &=& a_0 + a_1(x_1 - x_0) & = f(x_1) \\ P(x_2) &=& a_0 + a_1(x_2 - x_0) + a_2(x_2 - x_1)(x_2 - x_0) & = f(x_2) \\ \vdots & & \vdots & \vdots \end{array}$$

sukzessive berechnen lassen (dieses LGS hat Linksdreiecksgestalt!).

Definition 2.13 Der führende Koeffizient a_n des Interpolationspolynoms

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

von f zu den Knoten $x_0 \leq x_1 \leq \ldots \leq x_n$ 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.

Satz 2.14 Für jede Funktion f $\in C^n(\mathbb{R})$ und Knoten $x_0 \leq \ldots \leq x_n \in \mathbb{R}$ 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 $x_0, \ldots, x_n$.
Gilt darüber hinaus $f \in C^{n+1}(\mathbb{R})$, so folgt:

$$f(x) = P(x) + [x_0, \ldots, x_n, x]f \cdot \omega_{n+1}(x)\,. \tag{2.7}$$

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

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

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 (2.7). ■

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

Lemma 2.15 *i)* 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) Für zusammenfallende Knoten $x_0 = \ldots = x_n$ gilt

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

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

$$P(f|x_0, \ldots, x_n) = \frac{(x_i - x)\overbrace{P(f|x_0, \ldots, \widehat{x}_i, \ldots, x_n)}^{[x_0,\ldots,\widehat{x}_i,\ldots,x_n]f\, x^{n-1}+\mathbb{P}_{n-2}} - (x_k - x)P(f|x_0, \ldots, \widehat{x}_k, \ldots, x_n)}{x_i - x_k}, \tag{2.8}$$

was sich durch Überprüfen der Interpolationseigenschaft zeigen lässt mittels Einsetzen der Definitionen. Aus der Eindeutigkeit des führenden Koeffizienten folgt aus (2.8) unmittelbar Behauptung *i)*. Stimmen dagegen alle Knoten $x_0, \ldots, x_n$ überein, so ist das Interpolationspolynom

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

wie man durch Einsetzen in μ_i (vgl. oben) leicht einsieht. Sei nun $0 \leq k \leq n$, dann folgt:

$$\mu_k(P) = f^{(k)}(x_0) + \frac{k!(x - x_0)^1}{(k+1)!} f^{(k+1)}(x_0) + \ldots + \frac{k!(x - x_0)^{n-k}}{n!} f^{(n)}(x_0).$$

Somit gilt auch *ii)*. ■

Satz 2.16 Es sei $f \in C^n[a,b]$, $f^{(n+1)}(x)$ existiere für alle $x \in (a,b)$; weiterhin gelte $a \leq x_0 \leq x_1 \leq \ldots \leq x_n \leq b$. Dann gilt:

$$f(x) - P_n(f|x_0,\ldots,x_n)(x) = \frac{(x-x_0)\cdot\ldots\cdot(x-x_n)}{(n+1)!} f^{(n+1)}(\xi), \tag{2.9}$$

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

Beweis. Nach Konstruktion von $P_n(f|x_0,\ldots,x_n)$ gilt: $P_n(f|x_0,\ldots,x_n) = f(x_k)$ $(k = 0,1,\ldots,n)$. Es sei nun x fest und dabei ungleich $x_0,x_1,\ldots,x_n$. Ferner sei

$$K(x) := \frac{f(x) - P_n(f|x_0,\ldots,x_n)(x)}{(x-x_0)\cdot\ldots\cdot(x-x_n)}. \tag{2.10}$$

Nun betrachten wir die Funktion

$$W(t) := f(t) - P_n(f|x_0,\ldots,x_n)(t) - (t-x_0)\cdot\ldots\cdot(t-x_n)K(x). \tag{2.11}$$

Die Funktion $W(t)$ verschwindet also an den Stellen $t = x_0,\ldots,t = x_n$ und durch (2.10) auch an der Stelle $t = x$. Nach dem verallgemeinerten Satz von *Rolle*[2] verschwindet die Funktion $W^{(n+1)}(t)$ an einer Stelle ξ mit $\min\{x,x_0,\ldots,x_n\} < \xi < \max\{x,x_0,\ldots,x_n\}$. Das $(n+1)$-fache Differenzieren von (2.11) 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

$$K(x) = \frac{1}{(n+1)!} f^{(n+1)}(\xi). \tag{2.12}$$

Nach Einsetzen von (2.12) in (2.11) erhalten wir die Behauptung für $t = x$, da x Nullstelle von W ist. ■

Bemerkung 2.17 Im Beweis zum Approximationsfehler (vgl. Satz 4, (2.9)) haben wir gezeigt

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

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

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

Somit folgern wir:

[2] Rolle, Michel (1652-1719)

Für alle Knoten $x_0 \leq \ldots \leq x_n$ existiert ein $\xi \in [x_0, x_n]$, sodass gilt:

$$[x_0, \ldots, x_n]f = \frac{f^{(n)}(\xi)}{n!} \tag{2.13}$$

Die Auswertung der Rekursionsformel (2.8) 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.

■ **Beispiel 2.18** Wenn wir das Schema auf gegebene Daten (x_i, f_i) $(i = 0, \ldots, 3)$ anwenden, so erhalten wir

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

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-Funktion: 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. 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 2.19** Betrachten wir nun noch abschließend das Schema für die nichttriviale *Hermite*-Interpolationsaufgabe (d.h. auch Ableitungen werden interpoliert). Unter Beachtung

von Lemma 2.15, insbesondere (2.8) ergibt sich:

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

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 2.20 Es sei $\omega(x) = (x-x_0)\cdot\ldots\cdot(x-x_n)$ und $L_k(x)$ seien die *Lagrange*-Polynome zu den Knoten $x_0, \ldots, x_n$. Dann hat

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

die Interpolationseigenschaften

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

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

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

sodass wir nun Folgendes erhalten:

$$P'(x_\ell) = f(x_\ell)\Big(2L'_\ell(x_\ell) - \frac{\omega''(x_\ell)}{\omega'(x_\ell)}\Big) + f'(x_\ell).$$

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

$$\omega(x) = L_\ell(x)(x - x_\ell)\omega'(x_\ell)$$

nach zweimaligem Differenzieren

$$\omega''(x) = L''_\ell(x)(x - x_\ell)\omega'(x_\ell) + 2L'_\ell(x)\omega'(x_\ell).$$

Damit gilt an der Stelle x_ℓ

$$\frac{\omega''(x_\ell)}{\omega'(x_\ell)} = 2L'_\ell(x_\ell)$$

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

2.3. *Tschebyscheff*-Interpolation

Wie das folgende beispiel zeigen wird, hat die Verteilung der Stützstellen $x_0, \ldots, x_n$ über das Interpolationsintervall entscheidenden Einfluss auf die Güte der Approximation. Ein klassisches Beispiel hierfür stammt von *Runge*[3]:

Die Interpolationspolynome $P(f|x_1, \ldots, x_n)$ 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.

MATLAB-Beispiel:

Das Interpolationspolynom zu 12 äquidistanten Stützstellen und Stützwerten zur Funktion

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

ist in Abb. 2.1 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

[3] Runge, Carl (1856-1927)

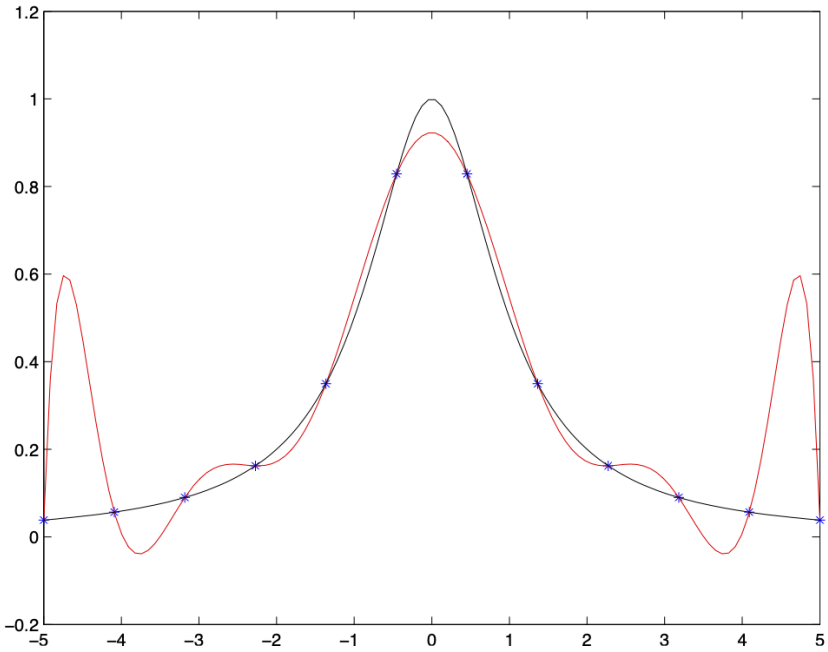


Abbildung 2.1.: Beispiel von Runge, d.h. Interpolationspolynom (schwarz) zur Funktion $f(x) = \frac{1}{1+x^2}$ (rot) im Intervall $I := [-5, 5]$ bei 12 äquidistanten Stützstellen.

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

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

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 $x_0, \ldots x_n$.

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

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

d.h. die Umkehrabbildung lautet:

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

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

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

so stellt $\widehat{P}(t) := P(x(t))$ die Lösung des ursprünglichen Problems mit führendem Koeffizienten $2^n/(b-a)^n$ dar. Für $x \in [-1,1]$ definieren wir die *Tschebyscheff*-Polynome durch (vgl. hierzu auch Kapitel 6.5):

$$T_n(x) = \cos(n \arccos(x)), \quad x \in [-1,1] \tag{2.14}$$

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

$$T_0(x) = 1, T_1(x) = x, T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x), \ k \geq 2 \tag{2.15}$$

Wir benötigen im Folgenden die schon in Kapitel 5 diskutierten Eigenschaften der *Tschebyscheff* Polynome, die wir der Einfachheit halber hier nochmals wiedergeben:

Bemerkung 2.21 (*i*) Der führende Koeffizient von T_n ist $a_n = 2^{n-1}$ $(n \geq 1)$
(*ii*) $|T_n(x)| \leq 1$ für $x \in [-1,1]$
(*iii*) Die Nullstellen von $T_n(x)$ sind

$$x_k := \cos(\frac{2k-1}{2n}\pi), \quad (k = 1, \ldots, n)$$

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

$$|T_n(x)| = 1 \Leftrightarrow x = \overline{x}_k = \cos(\frac{k\pi}{n}) \ \ mit\ k = 0, \ldots, n.$$

Satz 2.22 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 $T_n(x)$ minimal bezüglich der Maximumsnorm $\|f\|_\infty = \max_{x\in[-1,1]} |f(x)|$ 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 $|P(x)| < 1$ für $x \in [-1,1]$. 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 $\overline{x}_k := \cos(\frac{k\pi}{n})$ gilt:

$$T_n(\overline{x}_{2k}) = 1, \quad P_n(\overline{x}_{2k}) < 1 \quad \Rightarrow P_n(\overline{x}_{2k}) - T_n(\overline{x}_{2k}) < 0$$

$$T_n(\overline{x}_{2k+1}) = -1, \quad P_n(\overline{x}_{2k+1}) > -1 \quad \Rightarrow P_n(\overline{x}_{2k+1}) - T_n(\overline{x}_{2k+1}) > 0,$$

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 $x \in [-1, 1]$ geben derart, dass $|P_n(x)| \geq 1$ 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. ■

3. Splines

In diesem Kapitel wird die Theorie polynomialer und rationaler Splinefunktionen dargestellt.

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.

Definition 3.1 — Splineraum $\mathcal{S}^k(\mathcal{T})$. Es sei $\mathcal{T} = \{t_0, \ldots, t_{n+1}\}$ eine Knotenfolge von $n + 2$ paarweise verschiedenen Knoten

$$a = t_0 < \ldots < t_{n+1} = b\,.$$

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 $[t_j, t_{j+1}]$, $j = 0, \ldots, n$

$$s\big|_{[t_k,t_{k+1}]} \in \mathbb{P}_k$$

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 der Raum der stückweise stetigen Funktionen zu verstehen, d.h. unstetig nur an den Knoten x_j $(j = 0, \ldots, n + 1)$.

3.1. Kubische Spline-Interpolation

Im Folgenden betrachten wir die Interpolation mit **kubischen Splines**.

In vielen grafischen Anwendungen genügt die Glattheitsanforderung an die Interpolationsfunktion, dass die Krümmung stetig ist bzw. sie vom Auge als „glatt“ empfunden wird und die Randbedingungen symmetrisch gewählt werden können, d.h. an jedem Ende von gleicher Anzahl.

Dies ist einer der Hauptgründe, weswegen wir uns zuerst auf Funktionen aus $\mathcal{S}^k(\mathcal{T}) \subset C^2$ beschränken.

Untersuchen wir zuerst die Eigenschaften der kubischen Splines, bevor wir zu ihrer Konstruktion und Berechnung kommen.

Satz 3.2 Sei s ein interpolierender kubischer Spline zu der Funktion f an den Knoten $a = t_0 < \ldots < t_{n+1} = b$ und y eine beliebige interpolierende Funktion von f, sodass

$$s''(t) \cdot \big(y'(t) - s'(t)\big) = 0 \quad (t \in [a,b]). \tag{3.1}$$

Dann gilt

$$\|s''\|_2 := \Big(\int_a^b \big(s''(t)\big)^2\, dt\Big)^{1/2} \leq \|y''\|_2. \tag{3.2}$$

Beweis. Aus (3.2) 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 (3.1) und partieller Integration unter Berücksichtigung von $s(x)\big|_{[x_{i-1},x_i]} \in \mathbb{P}_3$ (und somit $s'''(x)\big|_{[x_{i-1},x_i]} \equiv c_i \in \mathbb{R}$) wiefolgt zeigen:

$$\begin{aligned}&\int_a^b s''(x)\big(y''(x) - s''(x)\big)dx = \sum_{i=1}^n \int_{x_{i-1}}^{x_i} s''(x)\big(y''(x) - s''(x)\big)dx \\ = &\sum_{i=1}^n \Bigg(\underbrace{s''(x)\big(y'(x) - s'(x)\big)}_{=\,0}\Big|_{x=x_{i-1}}^{x_i} - \int_{x_{i-1}}^{x_i} s'''(x)\big(y'(x) - s'(x)\big)dx\Bigg) \\ = &-\sum_{i=1}^n c_i \int_{x_{i-1}}^{x_i} y'(x) - s'(x)dx = -\sum_{i=1}^n c_i\Big[\big(y(x_i) - s(x_i)\big) - \big(y(x_{i-1}) - s(x_{i-1})\big)\Big] = 0\end{aligned}$$

wobei wir im letzten Schritt die Interpolationseigenschaft ausgenutzt haben. ■

Die Aussage dieses Satzes benötigen wir insbesondere zum Beweis des folgenden wichtigen Resultats über die Minimaleigenschaften der kubischen Splines.

Satz 3.3 — Minimaleigenschaft der kubischen Splines. Es sei $\mathcal{T} = \{x_i\}$ eine Knotenfolge mit $a = x_0 < \ldots < x_{n+1} = b$ 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 $s''(a) = s''(b)$ (falls f periodisch mit Periode $b-a$)	(periodische Randbedingung)

Ein solches $s \in \mathcal{S}^3(\mathcal{T})$ existiert und ist eindeutig bestimmt. Für jede interpolierende Funktion $y \in C^2[a,b]$, die diesselben Interpolations- und Randbedingungen erfüllt, gilt

ferner

$$\int_a^b (s''(x))^2 \, dx \leq \int_a^b (y''(x))^2 \, dx$$

Bevor wir diesen Satz beweisen noch eine Anmerkung zur Dimension von $\mathcal{S}^3(\mathcal{T})$ mit $\mathcal{T} = \{x_0, \ldots, x_{n+1}\}$. Für das Teilintervall $[x_i, x_{i+1}]$ der Länge $h_i := x_{i+1} - x_i$ wählen wir folgenden Ansatz:

$$s_i(x) := s(x)\big|_{[x_i, x_{i+1}]} = a_i(x - x_i)^3 + b_i(x - x_i)^2 + c_i(x - x_i) + d_i$$

Für seinen Wert und die Ableitungen s', s'' an den Endpunkten erhalten wir

$$\begin{aligned}
s_i(x_i) &= d_i = f(x_i) && (3.3)\\
s_i(x_{i+1}) &= a_i h_i^3 + b_i h_i^2 + c_i h_i + d_i = f(x_{i+1}) && (3.4)\\
s_i'(x_i) &= c_i && (3.5)\\
s_i'(x_{i+1}) &= 3a_i h_i^2 + 2b_i h_i + c_i && (3.6)\\
s_i''(x_i) &= 2b_i && (3.7)\\
s_i''(x_{i+1}) &= 6a_i h_i + 2b_i && (3.8)
\end{aligned}$$

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, sodass die Interpolationsbedingungen auf $[x_0, x_1]$ erfüllt sind. Durch die Interpolationsbedingungen und die Stetigkeit von s' und s'' an den Knoten folgt:

$$\begin{aligned}
s_1(x_1) &= d_1 = f(x_1) && (3.9)\\
s_1(x_2) &= a_1 h_1^3 + b_1 h_1^2 + c_1 h_1 + d_1 = f(x_2) && (3.10)\\
s_1'(x_1) &= c_1 = 3a_0 h_0^2 + 2b_0 h_0 + c_0 && (3.11)\\
s_1''(x_1) &= 2b_1 = 6a_0 h_0 + 2b_0 && (3.12)
\end{aligned}$$

Aus den Gleichungen (3.9) – (3.12) folgt die Eindeutigkeit der a_1, b_1, c_1, d_1. Per Induktion folgt dann auch die eindeutige Bestimmung der weiteren a_k, b_k, c_k, d_k, d.h. der Raum $\mathcal{S}^3(\mathcal{T})$ mit $\mathcal{T} = \{x_0, \ldots, x_{n+1}\}$ besitzt $(n+2) + 2 = n + 4$ Freiheitsgrade. Allgemein läßt sich zeigen:

$$dim\ \mathcal{S}^k(\mathcal{T}) = n + k + 1.$$

Beweis von Satz 3.3. 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 Splinefunktion $f \equiv 0$ der triviale Spline $s \equiv 0$ einzige Lösung ist.

Da $y \equiv 0$ alle Bedingungen erfüllt, folgt mit Satz (3.2), 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 $s = 0$. ■

Bemerkung 3.4 Der Satz 3.3 gilt unter Verallgemeinerung der Randbedingungen für beliebige Splineräume $\mathcal{S}^k$ $(k \geq 3)$. Siehe z.B. [7], Seite 252.

Berechnung kubischer Splines Kommen wir nun zur Berechnung kubischer Splines. Zusätzlich zu den Daten $y_i := f(x_i)$ $(i = 0, \ldots, n+1)$ ist es zweckmäßig Variablen $y_i'' := s_i''(x_i) = s_{i-1}''(x_i)$ einzuführen und die Variablen $a_i, \ldots, d_i$ durch diese zu ersetzen. Aus den Gleichungen (3.3), (3.4) sowie (3.7) und (3.8) gewinnt man das Gleichungssystem

$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ h_i^3 & h_i^2 & h_i & 1 \\ 0 & 2 & 0 & 0 \\ 6h_i & 2 & 0 & 0 \end{pmatrix} \begin{pmatrix} a_i \\ b_i \\ c_i \\ d_i \end{pmatrix} = \begin{pmatrix} s_i(x_i) \\ s_i(x_{i+1}) \\ s_i''(x_i) \\ s_i''(x_{i+1}) \end{pmatrix} = \begin{pmatrix} y_i \\ y_{i+1} \\ y_i'' \\ y_{i+1}'' \end{pmatrix}. \tag{3.13}$$

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 $a_i, \ldots, d_i$ und somit von s, falls s_i und s_i'' für alle $i = 0, \ldots, n+1$ bekannt sein sollten.

Das Auflösen von (3.13) nach a_i, b_i, c_i und d_i liefert nun

$$d_i = y_i \qquad b_i = \frac{1}{2} y_i'' \tag{3.14}$$

$$a_i = \frac{1}{6h_i}(y_{i+1}'' - y_i'') \qquad c_i = \frac{1}{h_i}(y_{i+1} - y_i) - \frac{1}{6} h_i (y_{i+1}'' + 2y_i''). \tag{3.15}$$

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 (3.21) ein, so erhalten wir

$$\begin{aligned} s_i'(x_{i+1}) &= 3h_i^2 a_i + 2h_i b_i + c_i \\ &= 3h_i^2 \big(\frac{1}{6h_i}(y_{i+1}'' - y_i'')\big) + 2h_i (\frac{1}{2} y_i'' + \frac{1}{h_i}(y_{i+1} - y_i) - \frac{1}{h_i}(y_{i+1}'' + 2y_i'') \\ &= h_i \big(\frac{1}{2} y_{i+1}'' - \frac{1}{2} y_i'' + y_i'' - \frac{1}{6} y_{i+1}'' - \frac{1}{3} y_i''\big) + \frac{1}{h_i}(y_{i+1} - y_i) \\ &= \frac{h_i}{6}(2y_{i+1}'' + y_i'') + \frac{1}{h_i}(y_{i+1} - y_i) \end{aligned}$$

bzw.

$$s_{i-1}'(x_i) = \frac{1}{h_{i-1}}(y_i - y_{i-1}) + \frac{h_{i-1}}{6}(2y_i'' + y_{i-1}'') \tag{3.16}$$

Die Stetigkeit von s' an den inneren Knoten liefert mit der letzten Gleichung (3.16) wegen der Darstellung von c_i aus (3.20) und (3.29) die Gleichung

$$\begin{aligned} &\frac{1}{h_{i-1}}(y_i - y_{i-1}) + \frac{1}{6} h_{i-1}(2y_i'' + y_{i-1}'') \\ = \; & s_{i-1}'(x_i) \stackrel{!}{=} s_i'(x_i) = c_i = \frac{1}{h_i}(y_{i+1} - y_i) - \frac{1}{6} h_i (y_{i+1}'' + 2y_i'') \end{aligned}$$

Sortieren von y_k'' nach links, y_k nach rechts und anschließende Multiplikation mit 6 liefert

$$h_{i-1}y_{i-1}'' + 2(h_{i-1}+h_i)y_i'' + h_i y_{i+1}'' = \frac{6}{h_i}(y_{i+1}-y_i) - \frac{6}{h_{i-1}}(y_i - y_{i-1}). \tag{3.17}$$

Diese Bedingung muss für alle inneren Knoten $x_1, \ldots, x_n$ erfüllt sein und liefert somit n Gleichungen für die Unbekannten $y_0'', \ldots, y_{n+1}''$.

Allerdings reichen die n Gleichungen für $n+2$ Unbekannte $y_0'', \ldots, y_{n+1}''$ nicht aus, um diese eindeutig zu bestimmen. Betrachten wir nun die unterschiedliche Behandlung der Randbedingungen.

Berücksichtigung natürlicher und vollständiger Randbedingungen Die vollständige Randbedingung $s'(a) = f'(a)$, bzw. $s'(b) = f'(b)$ liefert aufgrund von (3.20), (3.21) und (3.14) die weitere Gleichung

$$s'(a) = s_0'(x_0) = c_0 = \frac{1}{h_0}(y_1 - y_0) - \frac{1}{6}(y_1'' + 2y_0'') \stackrel{!}{=} f'(a).$$

Somit folgt

$$2h_0 y_0'' + h_0 y_1'' = \frac{6}{h_0}(y_1 - y_0) - 6f'(a), \tag{3.18}$$

bzw. aus

$$\begin{aligned} s'(b) &= s_n'(x_{n+1}) = 3a_n h_n^2 - 2b_n h_n + c_n \\ &= \frac{h_n}{2}(y_{n+1}'' - y_n'') + h_n y_n'' + \frac{1}{h_n}(y_{n+1} - y_n) - \frac{1}{6}h_n(y_{n+1}'' + 2y_n'') \\ &= h_n\left(\frac{1}{2}y_{n+1}'' - \frac{1}{2}y_n'' + y_n'' - \frac{1}{6}y_{n+1}'' - \frac{1}{3}y_n''\right) + \frac{1}{h_n}(y_{n+1} - y_n) \\ &= h_n\left(\frac{1}{3}y_{n+1}'' + \frac{1}{6}y_n\right) + \frac{1}{h_n}(y_{n+1} - y_n) \stackrel{!}{=} f'(b), \end{aligned}$$

für den rechten Rand

$$h_n y_n'' + 2h_n y_{n+1}'' = -\frac{6}{h_n}(y_{n+1} - y_n) + 6f'(b) \tag{3.19}$$

Für die natürlichen Randbedingung $y_0'' = y_{n+1}'' = 0$ ergeben sich folgende Gleichungen

$$y_0'' = 0 \tag{3.20}$$
$$y_{n+1}'' = 0 \tag{3.21}$$

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+1} gilt $y_{n+1} = 0$.

Im Falle $n = 5$ erhalten wir daraus folgendes lineares Gleichungssystem:

$$\begin{pmatrix} \star & \star & \star & \star & \star & \star \\ h_0 & 2(h_0+h_1) & h_1 & & & \\ & h_1 & 2(h_1+h_2) & h_2 & & \\ & & h_2 & 2(h_2+h_3) & h_3 & \\ & & & h_3 & 2(h_3+h_4) & h_4 \\ \star & \star & \star & \star & \star & \star \end{pmatrix} \cdot \begin{pmatrix} y_0'' \\ \vdots \\ \\ \vdots \\ y_5'' \end{pmatrix}$$

$$= \begin{pmatrix} & \star & \\ \frac{6}{h_1}(y_2-y_1) - \frac{6}{h_0}(y_1-y_0) \\ \frac{6}{h_2}(y_3-y_2) - \frac{6}{h_1}(y_2-y_1) \\ \frac{6}{h_3}(y_4-y_3) - \frac{6}{h_2}(y_3-y_2) \\ \frac{6}{h_4}(y_5-y_4) - \frac{6}{h_3}(y_4-y_3) \\ & \star & \end{pmatrix} \tag{3.22}$$

Die in diesem Schema mittels $\star$ gekennzeichnete erste und letzte Zeile ist dabei durch die Wahl der Randbedingungen (3.18), (3.22) bzw. (3.35), (3.36) gegeben.

Für vollständige Randbedingungen sind in (3.22) erste und letzte Zeile durch die Gleichungen (3.18) und (3.20) zu ersetzen, sodass wir erhalten:

$$\begin{pmatrix} 2h_0 & h_0 & & & \\ \star & \star & \star & & \\ & \ddots & & & \\ & & \star & \star & \star \\ & & & h_n & 2h_n \end{pmatrix} \begin{pmatrix} y_0'' \\ \star \\ \vdots \\ \star \\ y_{n+1}'' \end{pmatrix} = \begin{pmatrix} \frac{6}{h_0}(y_1-y_0) - 6f'(a) \\ \star \\ \vdots \\ \star \\ -\frac{6}{h_n}(y_{n+1}-y_n) + 6f'(b) \end{pmatrix}$$

Hierbei sind die mit $\star$ gekennzeichneten Zeilen 1 bis n dem System (3.22) zu entnehmen.

Analog erhalten wir für die Randbedingungen $y_0'' = f''(a)$, $y_{n+1}'' = f''(b)$

$$\begin{pmatrix} 1 & & & & \\ \star & \star & \star & & \\ & \ddots & & & \\ & & \star & \star & \star \\ & & & & 1 \end{pmatrix} \begin{pmatrix} y_0'' \\ \star \\ \vdots \\ \star \\ y_{n+1}'' \end{pmatrix} = \begin{pmatrix} f''(a) \\ \star \\ \vdots \\ \star \\ f''(b) \end{pmatrix}$$

Da im letzten Gleichungssystem y_0'' und y_{n+1}'' explizit bekannt sind, können wir es wie folgt reduzieren:

$$\begin{pmatrix} 2(h_0+h_1) & h_1 & & \\ h_1 & 2(h_1+h_2) & h_2 & \\ & & \ddots & \\ & & h_{n-1} & 2(h_{n-1}+h_n) \end{pmatrix} \cdot \begin{pmatrix} y_1'' \\ \vdots \\ \vdots \\ y_n'' \end{pmatrix} = \begin{pmatrix} \star & -h_0 y_0'' \\ & \star \\ & \star \\ \star & -h_n y_{n+1}'' \end{pmatrix}$$

Bemerkung 3.5 Man beachte, dass $y_0'' = c_0$ und $y_{n+1}'' = c_1$ 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.

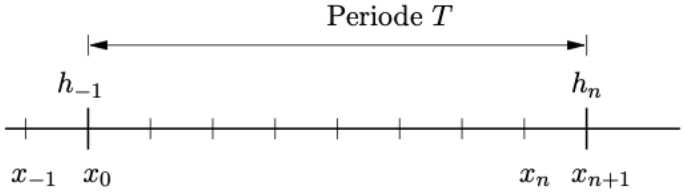
Berücksichtigung periodischer Randbedingungen Die Stützstellen $x_0 < \ldots < x_{n+1}$ seien nun so festgelegt, dass $x_{n+1} = x_0 + T$, wobei T die Periode der gesuchten Funktion darstelle.

Die periodischen Randbedingungen sind $f(x_0) = f(x_{n+1})$, $f'(x_0) = f'(x_{n+1})$ sowie $f''(x_0) = f''(x_{n+1})$. Mit den Größen $y_0 = y_{n+1}, y_1, \ldots, y_n$ und den Unbekannten $y_0'' = y_{n+1}'', y_1'', \ldots, y_n''$ ist die Stetigkeit an den $n+1$ Stützstellen $x_0, \ldots, x_n$ zu erfüllen (beachte, dass die Stetigkeit bei x_0 der bei x_{n+1} aufgrund der Periodizität entspricht).

Für die inneren Knoten $x_1, \ldots, x_n$ sind die Gleichungen gegeben durch (3.17). Aber auch zum Knoten x_0 sind die Gleichungen durch (3.17) gegeben, wenn man bei der Formulierung

$$\begin{aligned} h_{-1} &= x_0 - x_{-1} = x_{n+1} - x_n = h_n \\ y_{-1}'' &= y_n'' \quad \text{und } y_{-1} = y_n \end{aligned}$$

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



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 3.6 Der Name Spline läßt sich aus dem Englischen wie Latte oder Zeichenlineal übersetzen. Wieso die oben diskutierten Funktionen als Splines bezeichnet werden, erklärt sich durch eine physikalische Interpretation der Eigenschaft aus Satz 6:

Die 'Biegeenergie' E eines elastischen Körpers ist gegeben durch

$$E = \int_a^b \left(\frac{y''(x)}{(1+y'(x)^2)^{3/2}} \right)^2 dx,$$

also dem Integral über das Quadrat der Krümmung

$$\kappa(x) = \frac{y''(x)}{(1+y'(x)^2)^{3/2}}.$$

Für kleine $y'(x)$ liefert $y''(x)$ eine vernünftige Näherung an $\kappa(x)$, d.h. unter allen Interpolierenden 'minimiert' ein Spline das Integral über das Quadrat der Krümmung, also die 'glatteste' Interpolationsfunktion.

Die natürlichen Randbedingungen entsprechen also gerade der Situation, dass das Zeichenwerkzeug außerhalb des Interpolationsintervalls sich wie eine Gerade verhält.

Eine Matlab-Realisierung zur Berechnung der Koeffizienten a_i, b_i, c_i, d_i für alle Intervalls $[x_i, x_{i+1}]$ $(i = 0, \ldots, n)$ ist im Folgenden wieder gegeben.

MATLAB-Funktion: CoeffSpline.m

```
function coeff = CoeffSpline(t,y,kind,param)
%*** abhaengig von der Wahl von kind
% param(1,1)=f'(a) bzw. f''(a)
% param(2,1)=f'(b) bzw. f''(b)
t = t(:);
y = y(:);
n = size(t,1);
%*** Differenzvektoren
h = t(2:n)-t(1:n-1);
dy = y(2:end)-y(1:end-1);
%*** Initialisierung von A und b an den inneren Knoten
A = sparse(n,n);
A(2:n-1,:)=spdiags([h(1:end-1),2*(h(1:end-1)+h(2:end)), ...
                                          h(2:end)],0:2,n-2,n);
b = zeros(n,1);
b(2:n-1) = 6*(dy(2:end)./h(2:end)-dy(1:end-1)./h(1:end-1));
%*** Beruecksichtigung der Randbedingungen in A und b
switch kind
  case 'nat'
    A(1,1) = 1; b(1)   = param(1);
    A(n,n) = 1; b(end) = param(2);
```

```
22    case 'per'
23      if y(1) ~= y(end)
24        error('Data at first and last point must be the same!')
25      end
26      A(1,[1,2,n-1]) = [2*(h(1)+h(end)),h(1),h(end)];
27      A(n,[1,n]) = [1,-1];
28      b(1) = 6 * (dy(1)/h(1)-dy(end)/h(end));
29    case 'compl'
30      A(1,[1,2]) = [2*h(1),h(1)];
31      A(n,[n-1,n])=[h(end),2*h(end)];
32      b(1)=6 * (dy(1)/h(1) - param(1));
33      b(n)=6 * (param(2) - dy(end)/h(end));
34    otherwise
35      error('This kind does not exist!')
36  end
37  %*** Berechnung der y''
38  y2d=A\b;
39  %*** Berechnung der Koeffizienten a,b,c,d
40  coeff=[y(1:end-1),dy./h-h.*(y2d(2:end)+2*y2d(1:end-1))/6,...
41         y2d(1:end-1)/2,(y2d(2:end)-y2d(1:end-1))./(6*h)]';
```

3.2. Punktauswertung kubischer Splines

Nachdem wir nun im vorletzten Kapitel analysiert haben, wie sich aus den Daten $a = x_0 < x_1 < \ldots < x_{n+1} = b$ und $y_0, \ldots, y_{n+1}$ 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 k mit $x \in [x_k, x_{k+1}]$ bestimmen.

Bei der trivialen Realisierung, bei der man nacheinander $j = 0, 1, \ldots, n$ durchgeht und testet ob x in $[x_j, x_{j+1}]$ liegt, benötigt man bei einer äquidistanten Unterteilung $|x_{j+1} - x_j| =: h$ $(j = 0, \ldots, n)$ in $n + 1$ Teilintervalle durchschnittlich $n/2$ Abfragen. Da x mit der gleichen Wahrscheinlichkeit $1/(n + 1)$ in einem der Intervalle $[x_j, x_{j+1}]$ $(j = 0, \ldots, n)$ liegt, wird bei einem x, welches im letzten $\big((n + 1)$-ten$\big)$ Intervall $[x_n, x_{n+1}]$ oder vorletzten Intervall $[x_{n-1}, x_n]$ vorher n-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 $\frac{1}{n+1} + \frac{2}{n+1} + \ldots + \frac{n-1}{n+1} + \frac{n}{n+1} + \frac{n}{n+1} = \frac{(n+1)n}{2(n+1)} + \frac{n}{n+1} \approx \frac{n}{2} + 1$ Tests benötigt.

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 $[b, b]$ und nach s Bisektionen hat man das gesuchte Intervall gefunden. Die Anzahl der Tests ist für $2^{s-1} < n \leq 2^s$ gerade $s = \lceil \log_2 n \rceil$. Für $n = 100(10000)$ benötigt man durchschnittlich bei der sequentiellen Suche dem 501(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.
```

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 $t_j \in [x_{k_j}, x_{k_j+1}]$ $(k_j \in \{0,\ldots,n\})$, 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 fuer 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
```

MATLAB-Beispiel:

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.

```
>> 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.
```

3.3. Parametrisierte Kurven und Flächen

Implizite und parametrisierte Darstellung 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 Dartsellung 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 separat durch eine explizite Funktion eines unabhängigen Parameters dargestellt,

$$\mathbf{C}(t) = (x(t), y(t)) \quad a \leq t \leq b\,.$$

Somit ist $\mathbf{C}(t)$ eine vektorwertige Funktion des Parameters 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 Parameterdarstellung

$$x(t) = \cos(t),\ y(t) = \sin(t) \quad a \leq t \leq \pi/2\,.$$

Substituiert man $u = \tan(t/2)$ so erhält man die alternative Darstellung

$$x(u) = \frac{1 - u^2}{1 + u^2},\ y(u) = \frac{2u}{1 + u^2} \quad 0 \leq u \leq 1\,.$$

Die parametrische Darstellung ist folglich nicht eindeutig.

Beide Darstellungsformen haben Vor- und Nachteile, von denen einige hier genannt seien.

- 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, ob wohl 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 impliziter 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 erhält 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 mehrere wichtige 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

$$\mathbf{C}(t) = (x(t), y(t), z(t)) = \sum_{j=0}^{n} \mathbf{a}_j t^j \quad 0 \le t \le 1$$

mit $\mathbf{a}_j = (x_j, y_j, z_j)$. 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

```
1 function f = horner(a,t0)
2 % Compute point on a power basis curve
3 f = a(:,end);
```

```
4 for k = size(a,2)-1:-1:1
5   f = f.*t0+a(:,k);
6 end
```

3.4. Bernstein-Polynome und Bézier-Kurven

Die monomiale 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, nämlich die Bernstein-Polynome. Obwohl die parametrisierten Funktionen, dargestellt in monomialer Basis oder mit Bernstein-Polynomen, mathematisch äquivalent sind, so 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 3.7 — Bernstein-Polynom. Es seien $n \in \mathbb{N}_0$ und $0 \leq j \leq n$. Das j-te **Bernstein**-Polynom vom Grad n bezüglich des Intervalls $[0,1]$ ist das Polynom $B_j^n \in \mathbb{P}_n$ mit

$$B_j^n(t) = \binom{n}{j} t^j (1-t)^{n-j} . \tag{3.23}$$

(Zur Erinnerung $\binom{n}{j} = \frac{n!}{j!(n-i)!}$.)

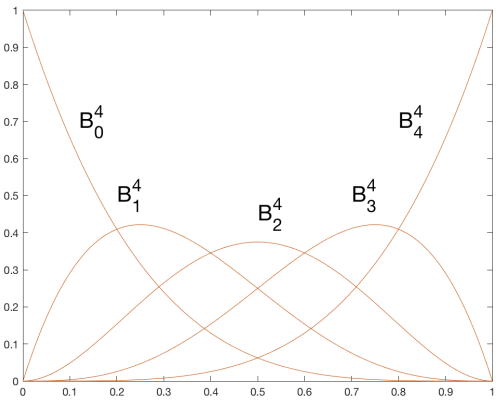


Abbildung 3.1.: Bernstein-Polynome $B_0^4, \ldots, B_0^4$ auf dem Intervall $[0,1]$.

Bemerkung 3.8 Es sei $n \geq 0$. Man beachte

$$\begin{aligned} 1 &= t + (1-t) = (t + (1-t))^n \\ &= \sum_{j=0}^{n} \binom{n}{j} t^j (1-t)^{n-j} = \sum_{j=0}^{n} B_j^n(t) . \end{aligned}$$

Satz 3.9 — Eigenschaften der Bernstein-Polynome. Die Bernstein-Polynome haben folgende Eigenschaften:

(i) $B_j^n(t) \geq 0$ für alle j, n und $0 \leq t \leq 1$ (Positivität).

(ii) $\sum_{j=0}^{n} B_j^n(t) = 1$ für alle $0 \leq t \leq 1$ (Zerlegung der Eins).

(iii) $B_0^n(0) = B_n^n(1) = 1$.

(iv) $B_j^n(t)$ hat genau ein Maximum im Intervall $[0, 1]$, und zwar bei $t = j/n$.

(v) $B_j^n(t) = B_{n-j}^n(1-t)$ für $j = 0, \ldots, n$ (Symmetrie).

(vi) $B_j^n(t) = (1-t)B_j^{n-j}(t) + tB_{j-1}^{n-1}(t)$ (Rekursionsformel);
wir definieren $B_j^n(t) \equiv 0$ für $j < 0$ oder $j > n$.

(vii)
$$\frac{d}{dt}B_i^n(t) = n\left(B_{i-1}^{n-1}(t) - B_i^{n-1}(t)\right)$$
mit $B_{-1}^{n-1}(t) \equiv B_n^{n-1}(t) \equiv 0$

Die Gleichung (3.23) liefert $B_0^0(t) = 1$. Aus der Eigenschaft 3.9.vi gewinnen wir die linearen Bernstein-Polynome

$$\begin{aligned} B_0^1(t) &= (1-t)B_0^0(t) + tB_{-1}^0(t) = 1 - t \\ B_1^1(t) &= (1-t)B_1^0(t) + tB_0^0(t) = t \end{aligned}$$

und quadratischen Bernstein-Polynome

$$\begin{aligned} B_0^2(t) &= (1-t)B_0^1(t) + tB_{-1}^1(t) = (1-t)^2 \\ B_1^2(t) &= (1-t)B_1^1(t) + tB_0^1(t) = 2t(1-t) \\ B_2^2(t) &= (1-t)B_2^1(t) + tB_1^1(t) = t^2\,. \end{aligned} \tag{3.24}$$

Die Rekursionsformel 3.9.vi liefert einen einfachen Algorihmus, um Werte der Bernstein-Polynome zu einem gegebenen t zu bestimmen. In Tabelle 3.1 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 3.2 dargestellt, effizient bestimmen.

Die Funktion `AllBernstein.m` kombiniert das in Tabelle 3.2 dargestellte Vorgehen mit Gleichung (3.23) um die Bernstein-Polynome n-ten Grades an einer gegeben Stelle t zu bestimmen.

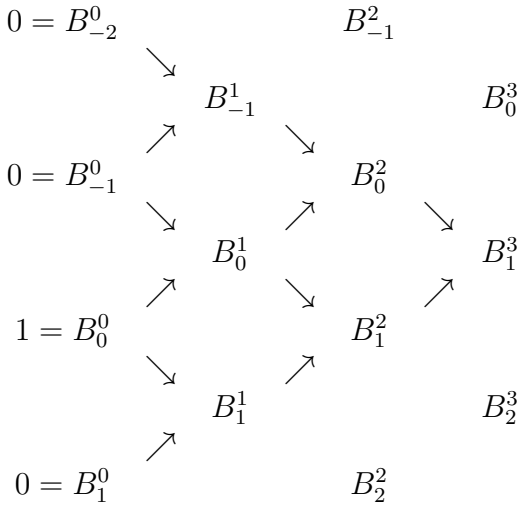


Tabelle 3.1.: Berechnung von B^3_1.

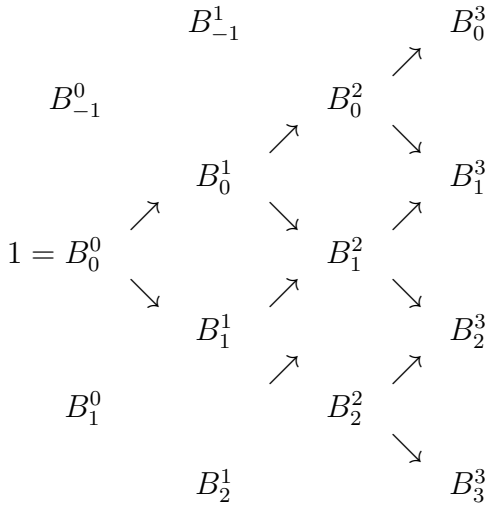


Tabelle 3.2.: Berechnung aller kubischen Bernstein-Polynome.

MATLAB-Funktion: AllBernstein.m

```
function B = AllBernstein(n,x)
%*** Compute all n-th Bernstein polynomials
B = zeros(n+1,1);
B(1) = 1;
for j=1:n
  for k=j:-1:1
    B(k+1) =  B(k+1) +  x * B(k);
    B(k)     = (1-x) * B(k);
  end
end
```

MATLAB-Beispiel:

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
```

Definition 3.10 — Bézier-Kurve. Gegeben seien $n+1$ **Kontrollpunkte** $\mathbf{P}_j \in \mathbb{R}^d$ ($d \in \mathbb{N}$). 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

$$\mathbf{C}(t) = \sum_{j=0}^{n} B_j^n(t)\, \mathbf{P}_j \,. \tag{3.25}$$

Bemerkung 3.11 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 $\mathbf{P}_j$. Diese Punkte heißen **Kontrollpunkte** zur Splinekurve s.

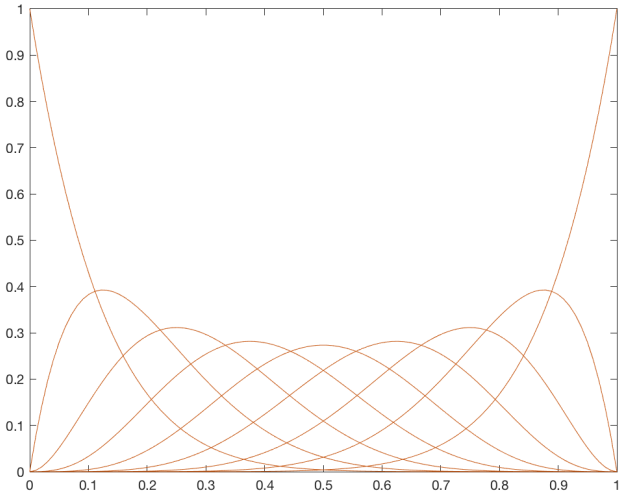


Abbildung 3.2.: Ergebnis der Matlab-Zeilen im o.g. Beispiel.

Bemerkung 3.12 Ist $d = 2$, so heißt die geradlinige Verbindung der Punkte $\mathbf{P}_0, \ldots, \mathbf{P}_n$ das **Kontrollpolygon**.

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

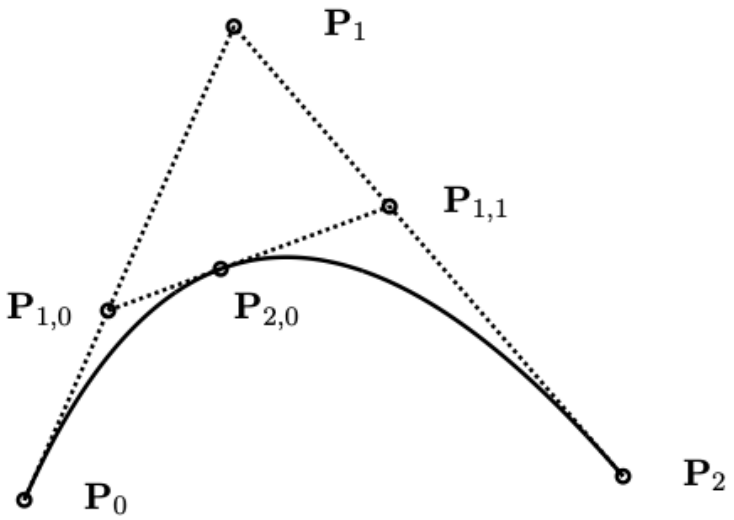


Abbildung 3.3.: Berechnung eines Punktes durch wiederholte lineare Interpolation für $t = 2/5$.

n-ten Grades mit $\mathbf{C}_n(P_0, \ldots, P_n)$, dann liefert uns die Rekursion 3.9.vi die Darstellung

$$\mathbf{C}_n(t; P_0, \ldots, P_n) = (1-t)\mathbf{C}_{n-1}(t; P_0, \ldots, P_{n-1}) + t\mathbf{C}_{n-1}(t; P_1, \ldots, P_n)\,.$$

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

$$\mathbf{P}_{k,j}(t_0) = (1-t_0)\mathbf{P}_{k-1,j}(t_0) + t_0\mathbf{P}_{k-1,j+1}(t_0) \quad \text{für} \begin{cases} k &= 1, \ldots, n \\ j &= 0, \ldots, n-k \end{cases} \tag{3.26}$$

Diese rekursive Berechnung von $\mathbf{C}(t_0) = \mathbf{P}_{n,0}(t_0)$ wird als **de Casteljau-Algorithmus** indexde Casteljau-Algorithmus bezeichnet.

MATLAB-Funktion: deCasteljau1.m

```
function C = deCasteljau1(P,t)
% Compute point on a bezier curve using Casteljau
%*** P = d x no of control points
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*:');
```

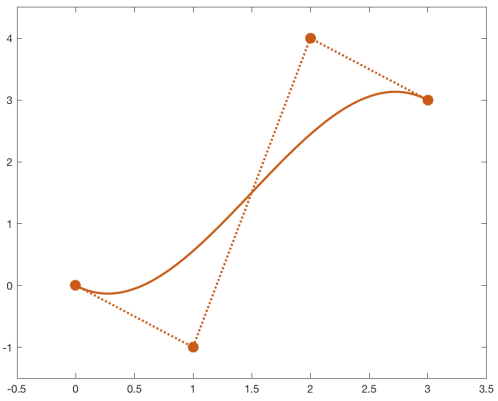


Abbildung 3.4.: Kontrollpolygon und Bézierkurve, Ergebnis des Matlab-Beispiels.

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 3.13** Es sei $n = 1$. Aus (3.24) erhalten wir $B_0^1(t) = 1 - t$ und $B_1^1(t) = t$. Die Darstellung (3.25) 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 3.14** Es sei $n = 2$. Aus (3.24) und (3.25) 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. 3.5 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 $\mathbf{P}_0, \mathbf{P}_1, \mathbf{P}_2$. ■

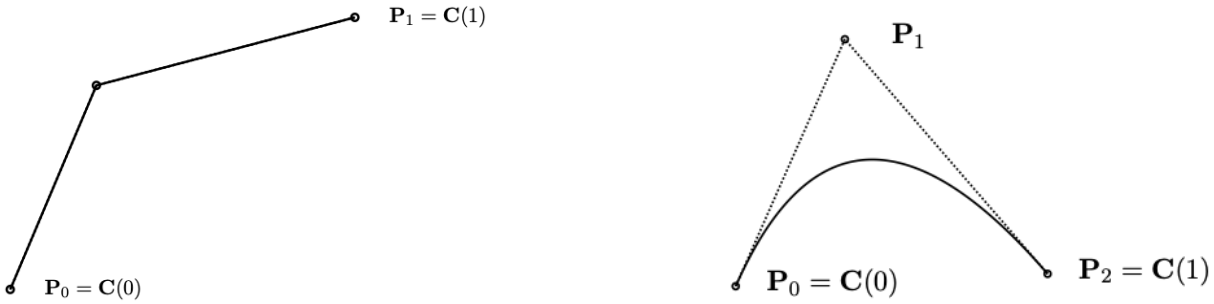


Abbildung 3.5.: Eine lineare (links) und quadratische (rechts) Bézierkurve.

■ **Beispiel 3.15** 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. 3.6 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 $\mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3$. ■

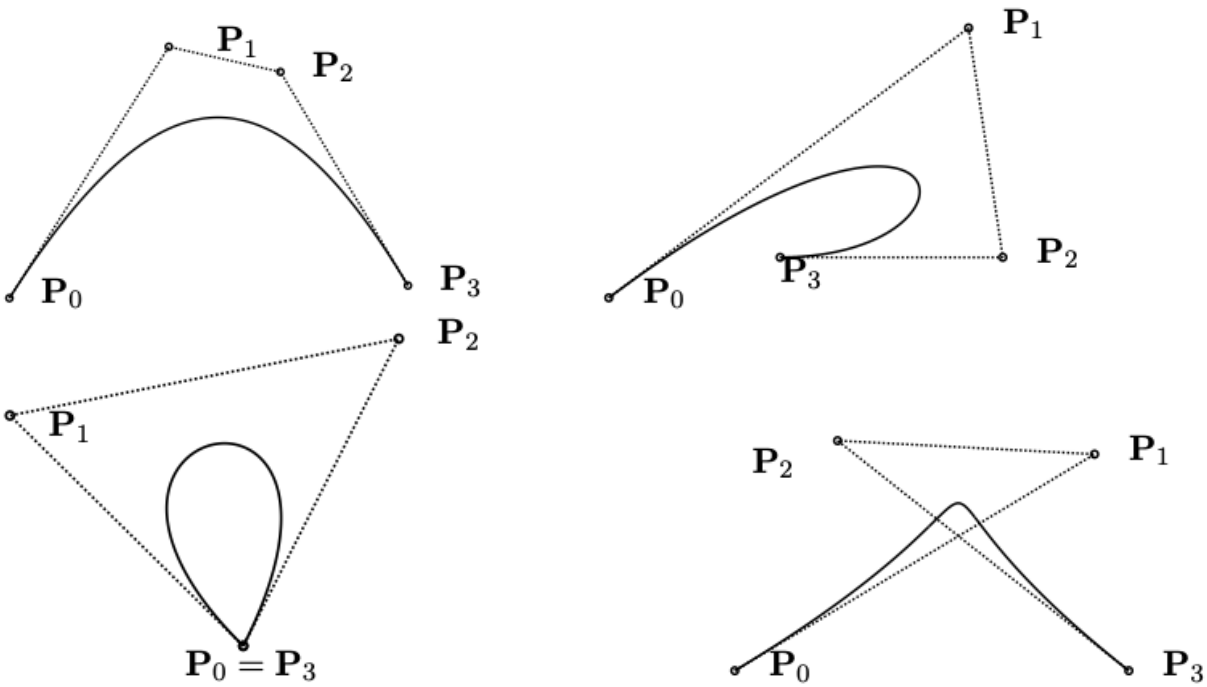


Abbildung 3.6.: Kubische Bézierkurve und zugehörige Kontrollpolygone.

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 von der st-Ebene in den 3-dimensionalen Raum dar, nämlich $\mathbf{S}(s,t) = (x(s,t), y(s,t), z(s,t)))$, $(s,t) \in R$. 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

$$\mathbf{b}_{ij} = (x_{ij}, y_{ij}, z_{ij}) \quad 0 \leq s,t \leq 1\,.$$

Man beachte, dass die Definitionsmenge dieser Abbildung das Quadrat $[0,1]^2$ ist. Verwendet man als Basisfunktionen wieder die Bernstein-Polynome so erhält man

$$\mathbf{S}(s,t) = \sum_{i=0}^{m} \sum_{j=0}^{n} B_i^m(s) B_j^n(t) \mathbf{P}_{ij} \quad 0 \leq s,t \leq 1\,. \tag{3.27}$$

Für ein festes s_0 gilt

$$\begin{aligned} \mathbf{C}_{s_0}(t) = \mathbf{S}(s_0,t) &= \sum_{i=0}^{m} \sum_{j=0}^{n} B_i^m(s_0) B_j^n(t) \mathbf{P}_{ij} \\ &= \sum_{j=0}^{n} B_j^n(t) \left(\sum_{i=0}^{m} B_i^m(s_0) \mathbf{P}_{ij} \right) \\ &= \sum_{j=0}^{n} B_j^n(t) \mathbf{Q}_j(s_0) \end{aligned} \tag{3.28}$$

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.

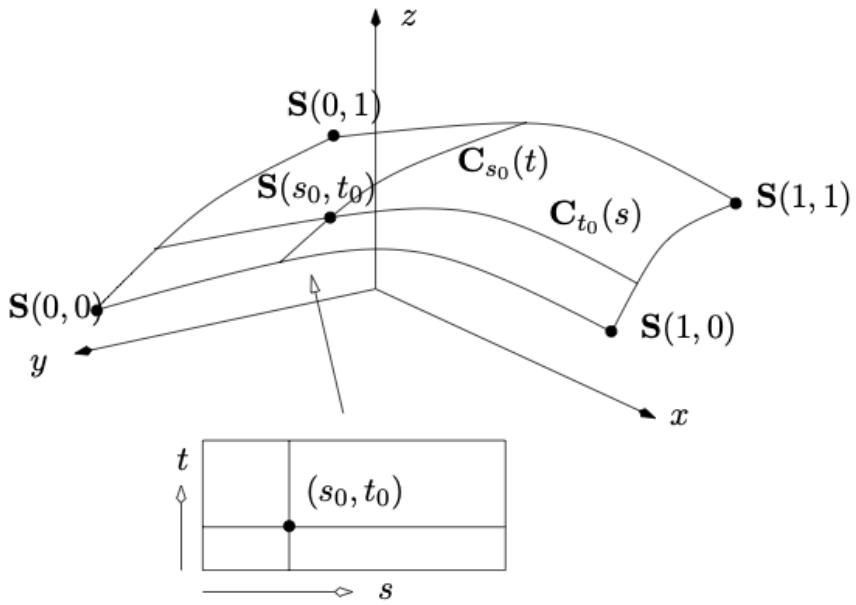


Abbildung 3.7.: Eine Tensorproduktfläche und isoparametrische Kurven.

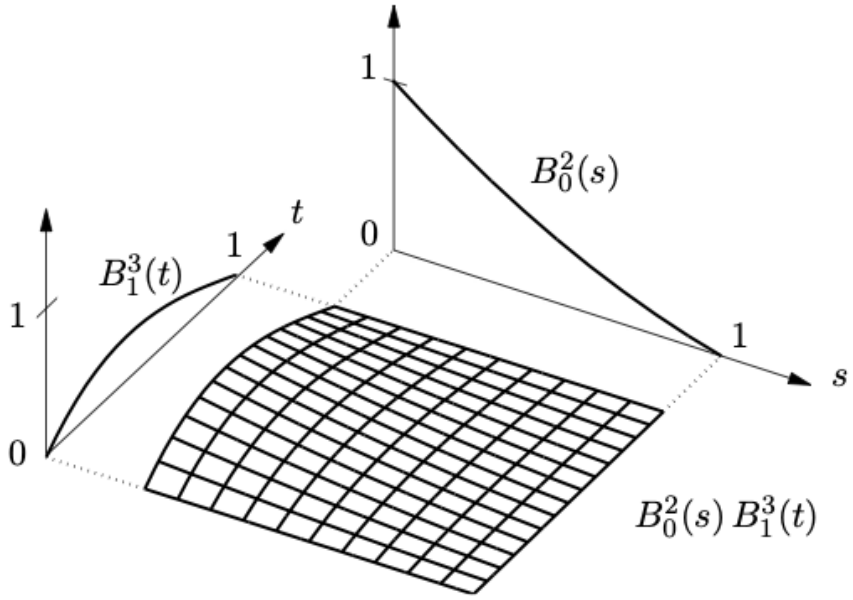


Abbildung 3.8.: Produkt von $B_0^2(s)$ und $B_1^3(t)$.

Mittels (3.28) kann man also (3.27) zu gegebenen (s_0, t_0) durch mehrmaliges anwenden des eindimensionalen deCasteljau-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. 3.9 ist das Kontrollnetz und die Bézierfläche beispielhaft dargestellt.

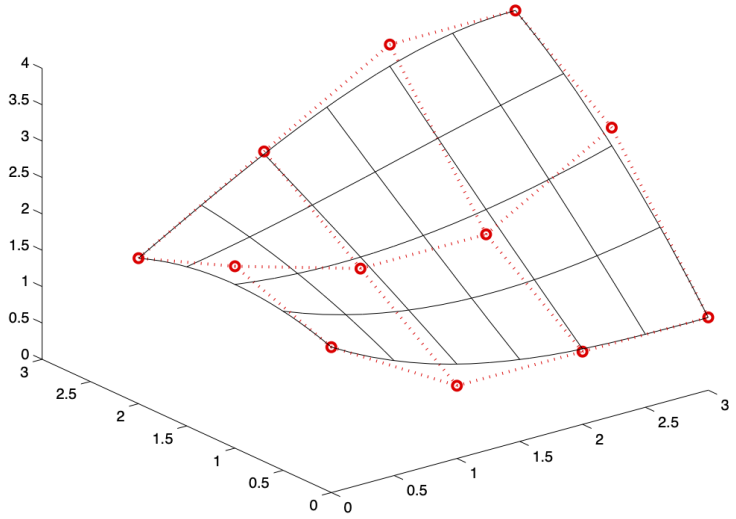


Abbildung 3.9.: Beispiel eines Kontrollnetz und Bézierfläche in $\mathbb{R}^2$.

3.5. B-Splines

In Kapitel 3.1 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[1] Ansatz beschäftigen, der durch die Anwendung in der Computergrafik motiviert ist. Wir werden eine weitere Basis 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 einer Funktion $f : \mathbb{R} \to \mathbb{R}$ die Menge $supp(f) := \overline{\{x \in \mathbb{R}, f(x) \neq 0\}}$ bezeichnet wird, wobei der Strich den Abschluß der Menge bezeichnet. Splines, die in dieser Basis dargestellt werden, nemmt man **B-Splines**.

Rekursive Definition der B-Splines-Basisfunktionen

Definition 3.16 — B-Splines-Basisfunktionen. Sei $\mathcal{T} = \{t_0, t_m\}$ eine nichtfallende Knotenfolge reeller Zahlen, d.h. $t_j \leq t_{j+1}$ $(j = 0, \ldots, m-1)$. Die t_j werden als **Knoten** und $\mathcal{T}$ als **Knotenvektor** bezeichnet. Die j-te B-Spline Basisfunktion vom Grade p (Ordnung $p+1$) ist definiert für $p = 0$ als stückweise konstante Funktion der Form

$$N_j^0(t) := \begin{cases} 1 & , \quad \text{falls } t_j \leq t < t_{j+1} \\ 0 & , \quad \text{sonst} \end{cases} \tag{3.29}$$

[1] Allgemeiner bzgl. des Polynomgrads auf jedem Teilintervall, als auch der Glattheitsanforderung an den Knoten zwischen den Teilintervallen, bzw. an den beiden Endpunkten.

und für $p > 0$ durch

$$N_j^p(t) := \frac{t - t_j}{t_{j+p} - t_j} N_j^{p-1}(t) + \frac{t_{j+p+1} - t}{t_{j+p+1} - t_{j+1}} N_{j+1}^{p-1}(t)\,. \tag{3.30}$$

Bemerkung 3.17
(i) N_j^0 ist eine Treppenfunktion, die auf dem halboffenen Intervall $[t_j, t_{j+1})$ Eins ist und sonst verschwindet.
(ii) Man beachte die rechtsseitige Stetigkeit in der Definition der N_j^0, d.h. $\lim_{t \to t_{j+1}^+} N_j^0(t) = N_{j+1}^0(t_{j+1})$.
(iii) Für $p > 0$ ist $N_j^p(t)$ eine Linearkombination von zwei Basisfunktionen vom Grade $(p-1)$.
(iv) Die Berechnung einer Menge von Basisfunktionen erfordert einen Knotenvektor $\mathcal{T}$ und einen Grad p.
(v) In (3.30) kann der Nenner im Bruch Null werden; dieser Quotient sei per Definition Null.
(vi) Die $N_j^p(t)$ sind stückweise polynomiale Funktionen auf der reellen Achse. Normalerweise ist nur das Intervall $[t_0, t_m]$ von Interesse.
(vii) Das j-te Knotenintervall $[t_j, t_{j+1})$ kann die Länge Null haben, da aufeinanderfolgende Knoten nicht verschieden sein müssen.
(viii) Die Berechnung der Basisfunktionen kann in dem bekannten Dreiecksschema erfolgen.
(ix) Die N_j^0 liefern auf dem Intervall $[t_0, t_m)$ eine Zerlegung der Eins und sind positiv.

Die rekursive Definition der B-Splines (3.30) liefert einen einfachen Algorihmus, um Werte der B-Splines zu einem gegebenen $t \in [t_j, t_{j+1})$ zu bestimmen.

MATLAB-Funktion: BSplineBasisFunc.m

```
function value = BsplineBasisFunc(U,t)
%*** given a node-vector,
%    computes the spline-function of maximal degree
  if t < U(1) | t > U(end)
    value = 0;
  else
    p = length(U)-2;
    if p<0
      error('Error. \nLenght of first argument must be at least
          two.');
    end
    N = double( t >= U(1:end-1)  &  t < U(2:end) );
    for k = 1 : p
      for j = 1 : (p - k + 1)
        if U(j+k) > U(j)
          left = ( t-U(j) ) / ( U(j+k) - U(j) );
```

```
16          else
17            left = 0.0;
18          end
19          if U(j+k+1) > U(j+1)
20            right = ( U(j+k+1) - t ) / ( U(j+k+1) - U(j+1) ) ;
21          else
22            right = 0.0;
23          end
24          N(j) = left * N(j) + right * N(j+1);
25        end
26      end
27      value = N(1);
28    end
29
30  end
```

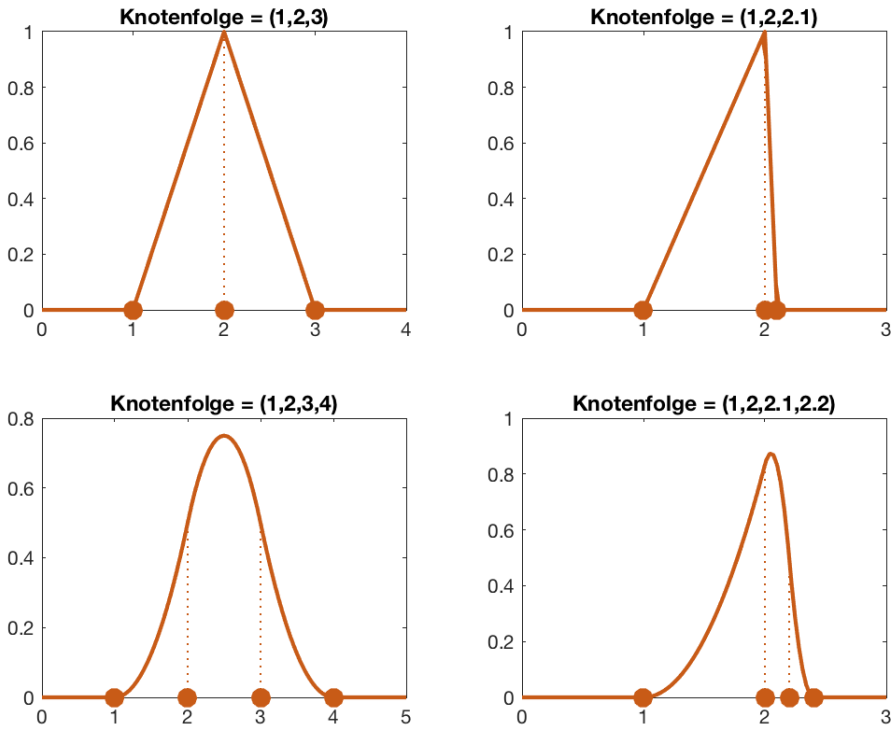


Abbildung 3.10.: Die Zerlegung von N_3^2 in seine stückweise polynomialen Teilfunktionen.

■ **Beispiel 3.18** Es sei $\mathcal{T} = \{t_0 = 0, t_1 = 0, t_2 = 0, t_3 = 1, t_4 = 1, t_5 = 1\}$ und $p = 2$. Für B-Spline-Basisfunktionen vom Grade 0, 1 und 2 lauten dann

$$\begin{aligned} N_0^0 &= N_1^0 = 0 \quad -\infty < t < \infty \\ N_2^0 &= \begin{cases} 1 & 0 \le t < 1 \\ 0 & \text{sonst} \end{cases} \\ N_3^0 &= N_4^0 = 0 \quad -\infty < t < \infty \end{aligned}$$

$$\begin{aligned} N_0^1 &= \frac{t-0}{0-0}N_0^0 + \frac{0-t}{0-0}N_1^0 = 0 && -\infty < t < \infty \\ N_1^1 &= \frac{t-0}{0-0}N_1^0 + \frac{1-t}{1-0}N_2^0 = \begin{cases} 1-t & 0 \le t < 1 \\ 0 & \text{sonst} \end{cases} \\ N_2^1 &= \frac{t-0}{1-0}N_2^0 + \frac{1-t}{1-1}N_3^0 = \begin{cases} t & 0 \le t < 1 \\ 0 & \text{sonst} \end{cases} \\ N_3^1 &= \frac{t-1}{1-1}N_3^0 + \frac{1-t}{1-1}N_4^0 = 0 && -\infty < t < \infty \end{aligned}$$

$$\begin{aligned} N_0^2 &= \frac{t-0}{0-0}N_0^1 + \frac{1-t}{1-0}N_1^1 = \begin{cases} (1-t)^2 & 0 \le t < 1 \\ 0 & \text{sonst} \end{cases} \\ N_1^2 &= \frac{t-0}{1-0}N_1^1 + \frac{1-t}{1-0}N_2^1 = \begin{cases} 2t(1-t) & 0 \le t < 1 \\ 0 & \text{sonst} \end{cases} \\ N_2^2 &= \frac{t-0}{1-0}N_2^1 + \frac{1-t}{1-1}N_3^1 = \begin{cases} t^2 & 0 \le t < 1 \\ 0 & \text{sonst} \end{cases} \end{aligned}$$

Man beachte, dass N_j^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

$$U = \{\underbrace{0, \ldots, 0}_{(p+1)-mal}, \underbrace{1, \ldots, 1}_{(p+1)-mal}\}$$

eine Verallgemeinerung der Bézier-Darstellung ist. ■

■ **Beispiel 3.19** Es sei $\mathcal{T} = \{t_0 = t_1 = t_2 = 0, t_3 = 1, t_4 = 2, t_5 = 3, t_6 = t_7 = 4, t_8 = t_9 = t_{10} = 5\}$ und $p = 2$. Für stückweise konstanten, linearen und quadratischen B-Spline-

Basisfunktionen lauten dann:

$$
\begin{aligned}
N_0^0 &= N_1^0 = 0 \quad -\infty < t < \infty \\
N_2^0 &= \begin{cases} 1 & 0 \le t < 1 \\ 0 & \text{sonst} \end{cases} \\
N_3^0 &= \begin{cases} 1 & 1 \le t < 2 \\ 0 & \text{sonst} \end{cases} \\
N_4^0 &= \begin{cases} 1 & 2 \le t < 3 \\ 0 & \text{sonst} \end{cases} \\
N_5^0 &= \begin{cases} 1 & 3 \le t < 4 \\ 0 & \text{sonst} \end{cases} \\
N_6^0 &= 0 \quad -\infty < t < \infty \\
N_7^0 &= \begin{cases} 1 & 4 \le t < 5 \\ 0 & \text{sonst} \end{cases} \\
N_8^0 &= N_9^0 = 0 \quad -\infty < t < \infty
\end{aligned}
$$

$$
\begin{aligned}
N_0^1 &= \frac{t-0}{0-0}N_0^0 + \frac{0-t}{0-0}N_1^0 = 0 && -\infty < t < \infty \\
N_1^1 &= \frac{t-0}{0-0}N_1^0 + \frac{1-t}{1-0}N_2^0 = \begin{cases} 1-t & 0 \le t < 1 \\ 0 & \text{sonst} \end{cases} \\
N_2^1 &= \frac{t-0}{1-0}N_2^0 + \frac{2-t}{2-1}N_3^0 = \begin{cases} t & 0 \le t < 1 \\ 2-t & 1 \le t < 2 \\ 0 & \text{sonst} \end{cases} \\
N_3^1 &= \frac{t-1}{2-1}N_3^0 + \frac{3-t}{3-2}N_4^0 = \begin{cases} t-1 & 1 \le t < 2 \\ 3-t & 2 \le t < 3 \\ 0 & \text{sonst} \end{cases} \\
N_4^1 &= \frac{t-2}{3-2}N_4^0 + \frac{4-t}{4-3}N_5^0 = \begin{cases} t-2 & 2 \le t < 3 \\ 4-t & 3 \le t < 4 \\ 0 & \text{sonst} \end{cases} \\
N_5^1 &= \frac{t-3}{4-3}N_5^0 + \frac{4-t}{4-4}N_6^0 = \begin{cases} t-3 & 3 \le t < 4 \\ 0 & \text{sonst} \end{cases} \\
N_6^1 &= \frac{t-4}{4-4}N_6^0 + \frac{5-t}{5-4}N_7^0 = \begin{cases} 5-t & 4 \le t < 5 \\ 0 & \text{sonst} \end{cases} \\
N_7^1 &= \frac{t-4}{5-4}N_7^0 + \frac{5-t}{5-5}N_8^0 = \begin{cases} t-4 & 4 \le t < 5 \\ 0 & \text{sonst} \end{cases} \\
N_8^1 &= \frac{t-5}{5-5}N_8^0 + \frac{5-t}{5-5}N_9^0 = 0 && -\infty < t < \infty
\end{aligned}
$$

Die folgenden N_j^2 sind bis auf die angegebenen Intervalle jeweils Null.

$$
\begin{aligned}
N_0^2 &= \frac{t-0}{0-0}N_0^1 + \frac{1-t}{1-0}N_1^1 &&= (1-t)^2 && 0 \le t < 1 \\
N_1^2 &= \frac{t-0}{1-0}N_1^1 + \frac{2-t}{2-0}N_2^1 &&= \begin{cases} 2t - \frac{3}{2}t^2 \\ \frac{1}{2}(2-t)^2 \end{cases} && \begin{matrix} 0 \le t < 1 \\ 1 \le t < 2 \end{matrix} \\
N_2^2 &= \frac{t-0}{2-0}N_2^1 + \frac{3-t}{3-1}N_3^1 &&= \begin{cases} \frac{1}{2}t^2 \\ -\frac{3}{2} + 3t - t^2 \\ \frac{1}{2}(3-t)^2 \end{cases} && \begin{matrix} 0 \le t < 1 \\ 1 \le t < 2 \\ 2 \le t < 3 \end{matrix} \\
N_3^2 &= \frac{t-1}{3-1}N_3^1 + \frac{4-t}{4-2}N_4^1 &&= \begin{cases} \frac{1}{2}(t-1)^2 \\ -\frac{11}{2} + 5t - t^2 \\ \frac{1}{2}(4-t)^2 \end{cases} && \begin{matrix} 1 \le t < 2 \\ 2 \le t < 3 \\ 3 \le t < 4 \end{matrix} \\
N_4^2 &= \frac{t-2}{4-2}N_4^1 + \frac{4-t}{4-3}N_5^1 &&= \begin{cases} \frac{1}{2}(t-2)^2 \\ -16 + 10t - \frac{3}{2}t^2 \end{cases} && \begin{matrix} 2 \le t < 3 \\ 3 \le t < 4 \end{matrix} \\
N_5^2 &= \frac{t-3}{4-3}N_5^1 + \frac{5-t}{5-4}N_6^1 &&= \begin{cases} (t-3)^2 \\ (5-t)^2 \end{cases} && \begin{matrix} 3 \le t < 4 \\ 4 \le t < 5 \end{matrix} \\
N_6^1 &= \frac{t-4}{5-4}N_6^1 + \frac{5-t}{5-4}N_7^1 &&= 2(t-4)(5-t) && 4 \le t < 5 \\
N_7^1 &= \frac{t-4}{5-4}N_7^1 + \frac{5-t}{5-5}N_8^1 &&= (t-4)^2 && 4 \le t < 5
\end{aligned}
$$

■

In Abbildung 3.11 ist die Zusammensetzung von N_3^2 aus den jeweiligen stückweise polynomialen Funktionen auf den einzelnen Teilintervallen grafisch dargestellt.

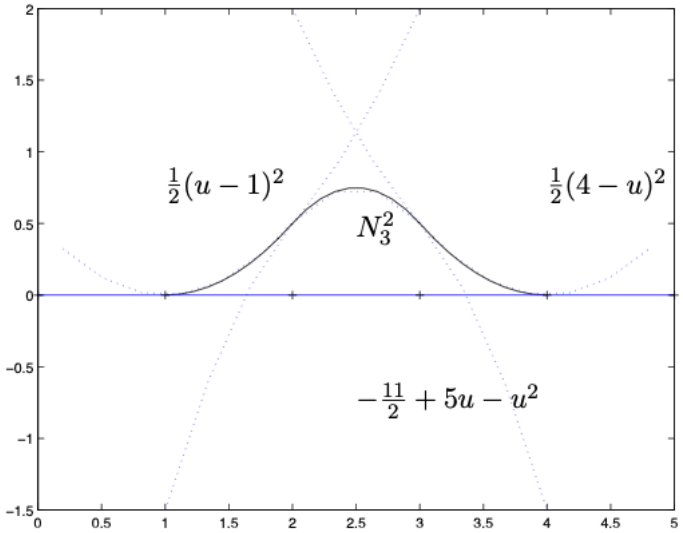


Abbildung 3.11.: Die Zerlegung von N_3^2 in seine stückweise polynomialen Teilfunktionen.

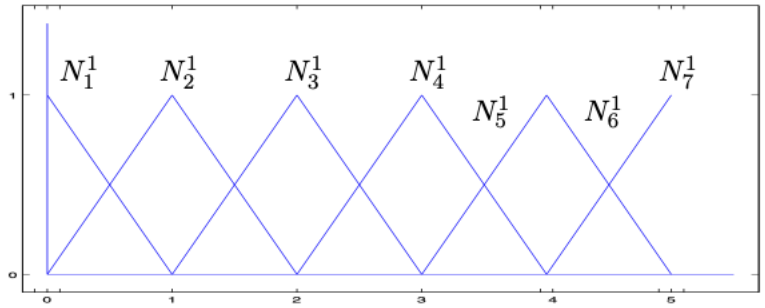


Abbildung 3.12.: Die stückweise linearen Basisfunktionen zu $\mathcal{T} = \{0, 0, 0, 1, 2, 3, 4, 4, 5, 5, 5\}$.

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_5^2 in Abbildung 3.13. Beachtet man die rekursive Definition der Basisfunktionen (3.30), 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 beüglich einer Basisfunktion.

Zum Beispiel hat $t = 0$ die Vielfachheit 3 im o.g. Knotenvektor $\mathcal{T}$, aber in Bezug auf die Basisfunktion N_1^2 if $t = 0$ ein Knoten mit der Vielfachheit 2. Die Basisfunktionen

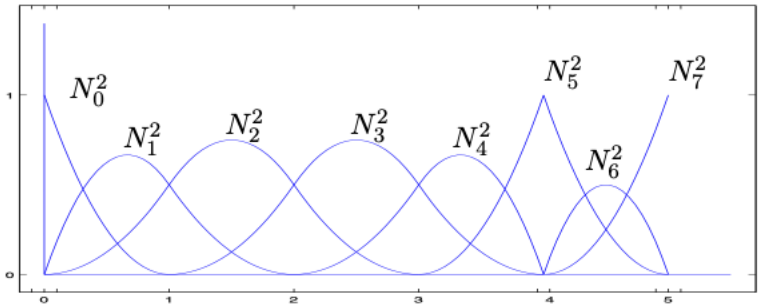


Abbildung 3.13.: Die stückweise quadratischen Basisfunktionen zu $\mathcal{T} = \{0, 0, 0, 1, 2, 3, 4, 4, 5, 5, 5\}$.

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 enthällt $t = 0$ gar nicht als Knoten.

Satz 3.20 Ist t_ℓ ein k-facher Knoten, d.h.

$$t_{\ell-1} < t_\ell = \ldots = t_{\ell+k-1} < t_{\ell+k},$$

so ist N_j^n an der Stelle t_ℓ mindestens $(n-k)$-mal stetig differenzierbar. Für die Ableitung von N_j^n gilt

$$\frac{d}{dt} N_j^n(t) = n \cdot \left(\frac{N_j^{n-1}(t)}{t_{j+n} - t_j} - \frac{N_{j+1}^{n-1}(t)}{t_{j+n+1} - t_{j+1}} \right)$$

Beweis. Der Beweis entnehme man [16]. ■

Definition 3.21 Sei $\mathcal{T}$ eine gegebene Knotenfolge. Wir bezeichnen den von den B-Splines B_{jk} bei vorgegebener Ordnung k aufgespannten Vektorraum mit $\mathcal{S}_{k,t}$.

Da zu einem $x \in [x_j, x_{j+1})$ höchstens die B-Splines $B_{j-k+1,k}, B_{j-k+2,k}, \ldots, B_{jk}$ von Null verschieden sind, läßt sich ein Element $s \in \mathcal{S}_{k,t}$ in der Form

$$s = \sum_{j \in \mathbb{Z}} a_j B_{jk}, \quad a_j \in \mathbb{R}$$

darstellen.

Satz 3.22 — *Marsdens²* Identiät. Bei gegebener Knotenfolge $\mathcal{T} := \{t_j\}, j \in \mathbb{Z}$, mit

$$\lim_{j \to \pm\infty} t_j = \pm\infty$$

sei für beliebiges $y \in \mathbb{R}$

$$\psi_{j1}(y) := 1, \quad \psi_{jk}(y) := (x_{j+1} - y)(x_{j+2} - y) \cdot \ldots \cdot (x_{j+k-1} - y), \quad k > 1.$$

Dann gilt

$$(x - y)^{k-1} = \sum_{j \in \mathbb{Z}} \psi_{jk}(y) B_{jk}(x)$$

Beweis. Sei $\big(a_j\big)_{j \in \mathbb{Z}}$ eine beliebige Folge reeller Zahlen. Dann folgt aus der Rekursion (3.30)

$$\sum_{j \in \mathbb{Z}} a_j B_{jk} = \sum_{j \in \mathbb{Z}} (a_{j-1}(1 - \omega_{jk} + a_j \omega_{jk}) B_{j,k-1}$$

Setzen wir $a_j := \psi_{jk}(x)$, so folgt

$$\begin{aligned} & a_{j-1}(1 - \omega_{jk}(x)) + a_k \omega_{jk}(x) \\ = \; & \Big(\big(x_j - y\big)\big(1 - \omega_{jk}(x)\big) + \big(x_{j+k-1} - y\big)\omega_{jk}(x) \Big) \psi_{j,k-1}(y) \\ = \; & (x - y)\psi_{j,k-1}(y). \end{aligned}$$

Damit gewinnen wir die Gleichung

$$\sum_{j \in \mathbb{Z}} \psi_{jk}(y) B_{jk}(x) = (x - y) \sum_{j \in \mathbb{Z}} \psi_{j,k-1}(y) B_{j,k-1}(x)$$

und somit mithilfe der Definitionen der ψ_{jk} und B_{j1}

$$\sum_{j \in \mathbb{Z}} \psi_{jk} B_{jk}(x) = (x - y)^{k-1} \sum_{j \in \mathbb{Z}} \psi_{j,1}(y) B_{j,1}(x) = (x - y)^{k-1}$$

■

Bemerkung 3.23 Da y im obigen Satz beliebig war, haben wir also gezeigt, dass $\mathbb{P}_{k-1} \subset \mathcal{S}_{k,t}$.

[5] benannt nach M. J. Marsden, vlg. hierzu [**Marsd**]

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 $t \in [t_j, t_{j+4}$.

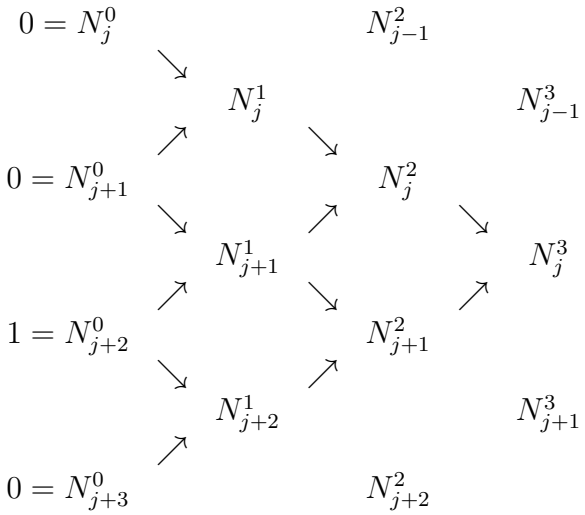


Tabelle 3.3.: N_j^3 ist nur auf dem Intervall $[t_j, t_{j+4})$ von Null verschieden.

In jedem Knotenintervall $[t_j, t_{j+1})$ sind maximal $p+1$ der N_i^p von Null verschieden, nämlich $N_{j-p}^p, \ldots, N_j^p$. 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 Grafik **??** dargestellt.

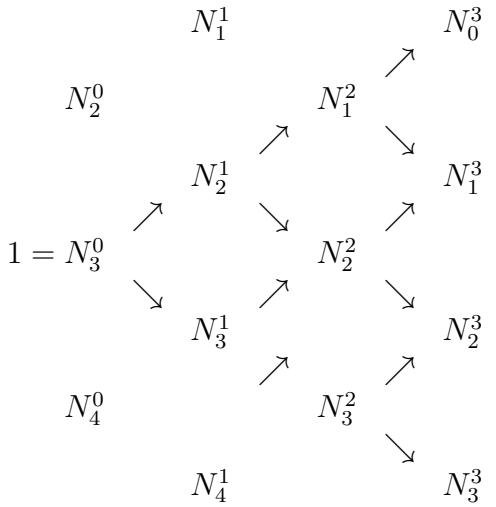


Tabelle 3.4.: N_3^0 ist nur auf dem Intervall $[t_3, t_4)$ von Null veschieden, Somit sind auch $N_0^3, \ldots, N_3^3$ die einzigen von Null verschiedenen kubischen Funktionen auf $[t_3, t_4)$.

MATLAB-Funktion: BasisFunc.m

```
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
```

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
```

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
```

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;
>> s = linspace(U(1),U(end),201);
>> for k = 1:length(s)
     C(:,k) = CurvePoint(p,U,P,s(k));
   end
>> plot(C(1,:),C(2,:),'-', ...
        P(1,:),P(2,:),'o:');
>> hold on
>> for j = 1 :25:length(s)
     V = CurveDerivs(p,U,P,s(j),1);
     quiver(V(1,1),V(2,1), ...
            0.2*V(1,2),0.2*V(2,2))
   end
```

3.6. Rationale B-Splines

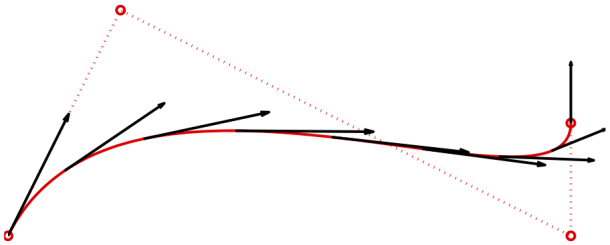


Abbildung 3.14.: Ergebnis der Matlab-Zeilen.

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
```

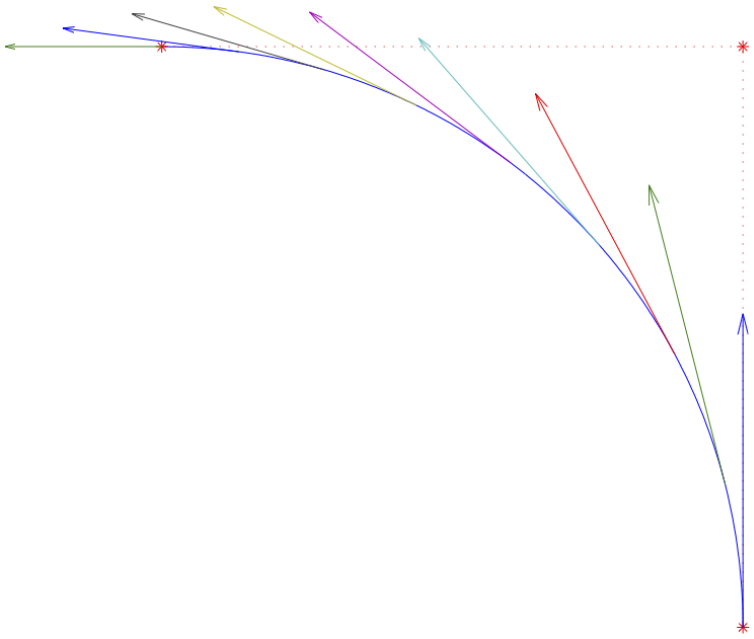


Abbildung 3.15.: Ergebnis der Matlab-Zeilen.

MATLAB-Beispiel:

Die folgenden Zeilen stellen die Bernstein-Polynome $B_0^8, \ldots, B_8^8$ graphisch dar.

```
>> w = [1, 1,  2];
>> P = [1, 1,  0;
        0  1,  1];
>> U = [0,0,0,1,1,1];   p=2;
>> s = linspace(0,1,201);
>> Pw = [P(1,:).*w;P(2,:).*w;w];
>> for k = 1:length(s)
     Cw(:,k) = RatCurvePoint(p,U,Pw,s(
        k));
   end
>> plot(Cw(1,:),Cw(2,:),'-', ...
        P(1,:),P(2,:),'*:');
>> hold on
>> for j = 1 :25:length(s)
     CK = RatCurveDerivs(p,U,Pw,s(j)
        ,1);
      quiver(CK(1,1),CK(2,1),0.3*CK
         (1,2),0.3*CK(2,2))
   end
```

3.7. 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);
```

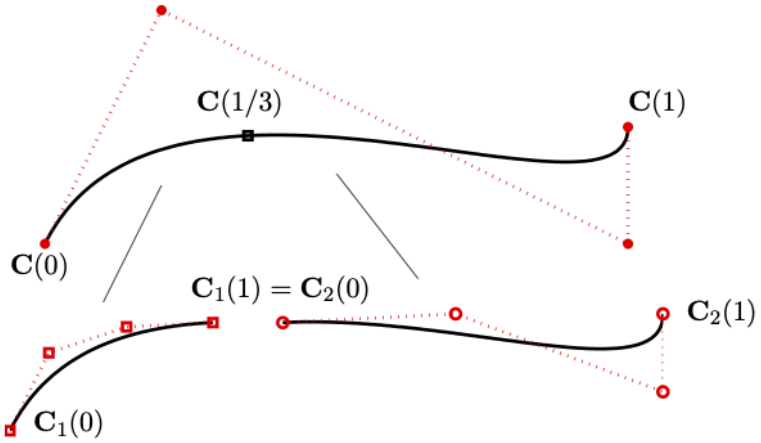


Abbildung 3.16.: Ergebnis der Matlab-Zeilen.

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
```

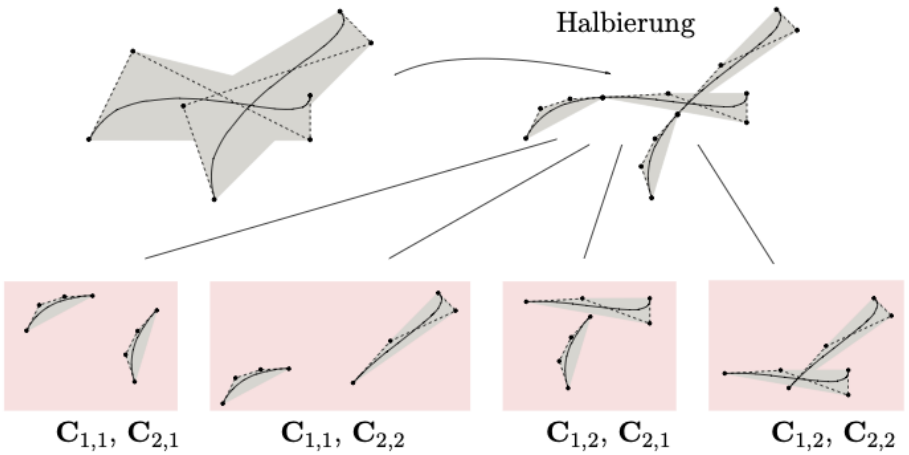


Abbildung 3.17.: Testen auf gemeinsamen Schnitt der einzelnen konvexen Hüllen nach Halbierung der einzelnen Bézierkurven $\mathbf{C}_1 = \mathbf{C}_{1,1} \cup \mathbf{C}_{1,2}$ und $\mathbf{C}_2 = \mathbf{C}_{2,1} \cup \mathbf{C}_{2,2}$.

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);
```

```
13      points = [points,CurveBisection(p1,U11,Q11,p2,U12,Q12,tol)
            ];
14      points = [points,CurveBisection(p1,U11,Q11,p2,U22,Q22,tol)
            ];
15      points = [points,CurveBisection(p1,U21,Q21,p2,U12,Q12,tol)
            ];
16      points = [points,CurveBisection(p1,U21,Q21,p2,U22,Q22,tol)
            ];
17    end
18  end
19
20  function h = myConvHull(Q,flag,a,b)
21  % Q
22  % [flag,a,b] = isLine(Q,1e-7,1e-7)
23
24  if flag
25    h = [a,b,a];
26  else
27    h = Q(:,convhull(Q(1,:),Q(2,:)));
28  end
```

MATLAB-Beispiel:

Die folgenden Zeilen stellen die Bernstein-Polynome $B_0^8, \ldots, B_8^8$ graphisch dar.

```
>> 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
```

3.8. B-Spline Interpolation

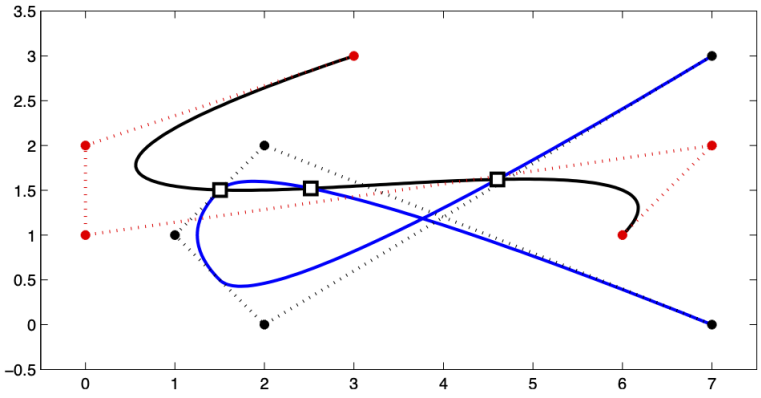


Abbildung 3.18.: Ergebnis der Matlab-Zeilen.

4. Numerische Quadratur

Nach dem Hauptsatz der Intergral- und Integralrechnung, existiert zu jeder stetigen Funktion eine Stammfunktion. Doch der Satz von Liouville (1835) besagt wiederum, dass nicht alle Stammfunktionen mit Hilfe der elementaren Funktionen (wie den ganzen rationalen Funktionen, den Wurzelfunktionen, den trigonometrischen Funktionen und ihren Umkehrfunktionen oder auch der Exponential- und der Logarithmusfunktion) dargestellt werden können. Dies trifft bereits auch schon so einfache Funktionen wie:

$$e^{-x^2}\,, \quad \frac{\sin x}{x}\,, \quad \frac{\log x}{x-1}\,, \quad x^x\,, \quad \sqrt{\cos x}\,, \quad \sqrt{\log x}\,, \quad \sqrt{1+x^{-4}}\,, \quad \sqrt[6]{1-x^{-7}}\,, \quad \frac{1}{\log x}\,.$$

Die numerische Quadratur (Quadratur $\approx$ veralterter Ausdruck für Flächenberechnung) 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 besser noch stückweise glatte Funktion).

Es ist nun evident, dass im Allgemeinen die Berechnung der Funktionswerte allein zu keiner gesicherten Aussage bezüglich $I(f)$ führt; vielmehr bedarf es einer weiteren zusätzlichen Information über f, welche die Funktionswerte an denjenigen Stellen, an denen keine Werte errechnet worden sind, eingrenzt. Diese - globale - Information nennen wir die *Co-Observation* des gegebenen Problems, und interpretieren diese *Co-Observation* als eine Menge $\mathcal{C}$ von Funktionen.

Es sei nun $\mathcal{C}$ fixiert, Q ein Algorithmus , $Q[f]$ der durch diesen bei der Anwendung auf f erzeugte Schätzwert für $I(f)$. Dann stellt

$$\varrho(Q;\mathcal{C}) := \sup_{f\in\ \mathcal{C}} \left| \int_a^b f(x)\ dx - Q[f] \right| \tag{4.1}$$

offenbar die durch Q in $\mathcal{C}$ garantierte Genauigkeit dar. Für einige $\mathcal{C}$ und Q werden wir $\varrho(Q;\mathcal{C})$ näher untersuchen.

[1] Riemann, Georg Friedrich Bernhard (1826-1866)

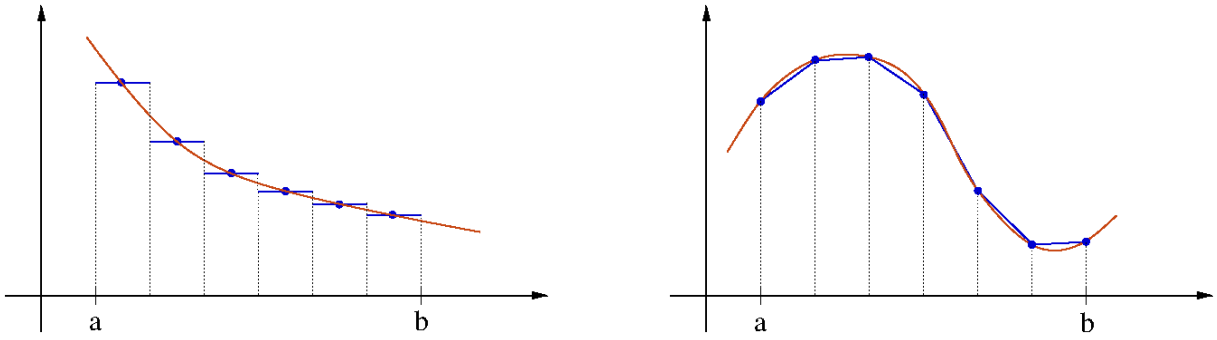


Abbildung 4.1.: Illustration der summierten Mittelpunktsformel (links) und summierten Trapezformel (rechts).

4.1. Mittelpunkt- und Trapezregel

Die folgende Grafik 4.1 kommt einem sicherlich bekannt vor, bei der das gesuchte Integral durch eine summierte Mittelpunkts- oder Trapezformel approximiert wird.

Betrachten wir dazu die Mittelpunkts- oder Trapezformel zuerst auf einem einzelnen Intervall.

Beginnen wir mit der sicherlich einfachsten Quadraturformel.

■ **Beispiel 4.1 — Mittelpunktsregel.** Wir setzen

$$\mathcal{C}_M^{(r)} := \left\{ f \;\middle|\; f^{(r)} \text{ stetig und } \sup_{a \le x \le b} |f^{(r)}(x)| \le M \right\}$$

für $r \in \mathbb{N}$ und $M > 0$, und betrachten nachfolgende Grafik 4.2:

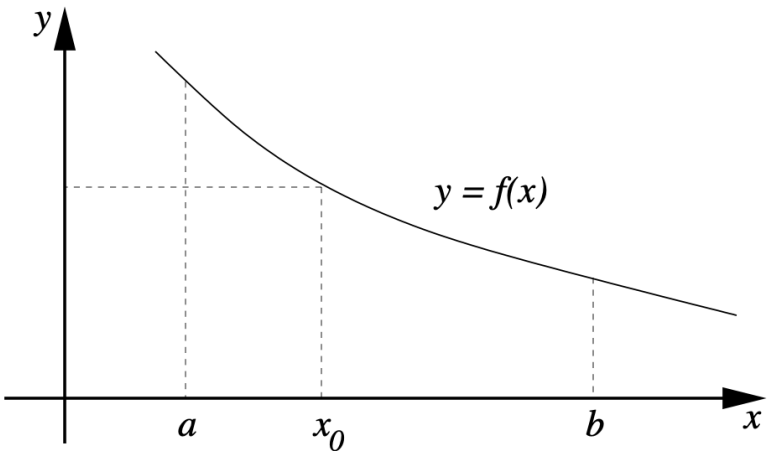


Abbildung 4.2.: Graph einer Funktion f.

Sei nun $x_0 \in (a, b)$ ersteinmal beliebig, dann gilt für $f \in C^1[a, b]$ (mit der uns bekannten Abschätzung des Interpolationsfehlers):

$$\int_a^b |f(x) - P(f|x_0)(x)|\, dx \;\le \sup_{a \le x \le b} \frac{|f'(x)|}{1!} \int_a^b |x - x_0|\, dx. \tag{4.2}$$

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 x_0 nun so wählen, dass $J(x_0)$ bzw. der rechte Teil der Ungleichung (4.2) 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 $J'' = 2 > 0$. Es gilt $J(\frac{a+b}{2}) = (\frac{a+b}{2})^2 - \frac{(a+b)^2}{2} + \frac{a^2+b^2}{2} = \frac{(b-a)^2}{4}$.

Damit haben wir folgende Quadraturformel **(Mittelpunktsformel)** motiviert:

$$Q^{Mi}[f] := (b-a) \cdot f\left(\frac{a+b}{2}\right) \tag{4.3}$$

Mit obiger Herleitung haben wir auch gezeigt, dass

$$\begin{aligned} \varrho(Q, \mathcal{C}_M^{(1)}) &\leq \frac{M}{1!} \cdot J\Big(\frac{a+b}{2}\Big) \\ &= M \cdot \Big(\frac{(a+b)^2}{4} - \frac{(a+b)^2}{2} + \frac{a^2+b^2}{2}\Big) \\ &= M \cdot \frac{(b-a)^2}{4} \end{aligned}$$

als Maß für die Genauigkeit von (4.7) gilt.

Bemerkung 4.2 Schaut man genauer hin, stellt man für $\alpha, \beta \in \mathbb{R}$ fest, dass gilt

$$\int_a^b \alpha x + \beta \, dx = \frac{\alpha}{2}x^2 + \beta x\Big|_{x=a}^b = (b-a)\left(\alpha x + \beta\Big|_{x=(a+b)/2}\right) = Q^{Mi}[\alpha x + \beta]\,,$$

d.h. die Mittelpunktsformel ist exakt für lineare Funktionen.

Für Funktionen $f \in C^2[a,b]$ können wir daher die Fehlabschätzung noch verbessern. Sei $p \in \mathbb{P}_1$ das eindeutige Interpolationspolynom an f mit $p(x_0) = f(x_0)$ und $p'(x_0) = f'(x_0)$ mit $x_0 = \frac{a+b}{2}$. Trivialerweise gilt $Q^{Mi}[f] = Q^{Mi}[p]$ und, da die Mittelpunktsformel für lineare Funktionen exakt ist, auch $Q^{Mi}[p] = I(p)$. Es folgt mit der uns bekannten Abschätzung für den hermiteschen Interpolationsfehler

$$\begin{aligned} &\big|I(f) - Q^{Mi}[f]\big| = \big|I(f) - Q^{Mi}[p]\big| = |I(f) - I(p)| = \left|\int_a^b f(x) - P(f|x_0,x_0)(x)\right| \\ = &\left|\int_a^b \frac{(x-x_0)^2}{2} f''(\xi(x))\, dx\right| \leq \sup_{a\leq x\leq b} |f''(x)| \int_a^b \frac{(x-x_0)^2}{2}\, dx = \frac{(b-a)^3}{24} \sup_{a\leq x\leq b} |f''(x)|\,. \end{aligned}$$

Wir fassen zusammen

Die **Mittelpunktsformel** ist exakt für lineare Funktionen und es gelten (unter entsprechenden Regularitätseigenschaften) die Abschätzungen

$$\varrho(Q^{Mi}, \mathcal{C}_M^{(1)}) \le M \cdot \frac{(b-a)^2}{4} \quad \text{und} \quad \varrho(Q^{Mi}, \mathcal{C}_M^{(2)}) \le M \cdot \frac{(b-a)^3}{24}\,. \tag{4.4}$$

∎

Zerlegt man das Intervall (a, b) in äquidistante Intervalle (x_k, x_{k+1}) $(k = 0, \ldots, n-1)$ mit $x_k = a + k\,(b-a)/n$ (wie in Grafik 4.1 links) und wendet für jedes Teilintegral in

$$\int_a^b f(x)\,dx = \sum_{k=0}^{n-1} \int_{x_k}^{x_{k+1}} f(x)\,dx \quad \approx \sum_{k=1}^{n} \frac{b-a}{n} f\left(a + (k-1/2)\,(b-a)/n\right)\,dx$$

die Mittelpunktsformel an, so erhalten wir die

summierte Mittelpunktsformel:

$$Q_n^{Mi}[f] := \frac{b-a}{n} \sum_{k=1}^{n} f\left(a + \left(k - \frac{1}{2}\right)\frac{b-a}{n}\right) \tag{4.5}$$

Für $f \in \mathcal{C}_M^{(1)}$ und jedes Teilintervall (x_k, x_{k+1}) $(k = 0, \ldots, n-1)$ gilt

$$\left| \int_{x_k}^{x_{k+1}} f(x) - f\left(\frac{x_k + x_{k+1}}{2}\right)\,dx \right| \le M \frac{(x_{k+1} - x_k)^2}{4}$$

und zusammenfassend für die **summierte Mittelpunktsformel**

$$\left| \int_a^b f(x) - Q_n^{Mi}[f]\,dx \right| \le M \sum_{k=0}^{n-1} \frac{(x_{k+1} - x_k)^2}{4} = M \cdot \frac{(b-a)^2}{4n}$$

Analoges erhält man auch für $f \in C^2[a, b]$.

Die **summierte Mittelpunktsformel** mit n Quadraturpunkten ist exakt für lineare Funktionen und es gelten (unter entsprechenden Regularitätseigenschaften) die Abschätzungen

$$\varrho(Q_n^{Mi}, \mathcal{C}_M^{(1)}) \le M \cdot \frac{(b-a)^2}{4\,n} \quad \text{und} \quad \varrho(Q_n^{Mi}, \mathcal{C}_M^{(2)}) \le M \cdot \frac{(b-a)^3}{24\,n^2}\,. \tag{4.6}$$

In Grafik 4.1 (rechts) haben wir eine weitere Quadraturformel motiviert, nämlich die

Trapezformel:

$$Q^{Tr}[f] := \frac{b-a}{2}\Big(f(a) + f(b)\Big)\,. \tag{4.7}$$

bzw.

summierte Trapezformel:

$$Q_n^{Tr}[f] := \frac{b-a}{n-1}\left(\frac{f(a)}{2} + \sum_{k=1}^{n-2} f\left(a + k\frac{b-a}{n-1}\right) + \frac{f(b)}{2}\right) \tag{4.8}$$

Eine Analyse wie zuvor liefert für $f \in C^2[a,b]$ und

die **Trapezformel** bzw. die **summierte Trapezformel** mit n Quadraturpunkten

$$\varrho(Q^{Trapez}, \mathcal{C}_M^{(2)}) \leq M \cdot \frac{(b-a)^3}{12}, \quad \varrho(Q_n^{Trapez}, \mathcal{C}_M^{(2)}) \leq M \cdot \frac{(b-a)^3}{12(n-1)^2},$$

Inwiefern läßt sich die Frage beantworten, ob eine Quadraturformel nun einen optimalen Algorithmus zur numerischen Berechnung beliebiger $I(f)$ darstellt. Dazu zitieren wir zwei klassische Ergebnisse

Es sei ω eine auf $I := [0, b-a]$ definierte monoton-wachsende Funktion, die $w(0) = 0$ und $w(x+y) \leq w(x) + w(y)$ für $x, y \in I$ erfülle. Der Stetigkeitsmodul w ist ein Begriff aus dem Gebiet der mathematischen Analysis, welcher dazu genutzt wird, einen Zusammenhang zwischen der Glattheit einer Funktion und der Approximationsgeschwindigkeit bei der Approximation durch Polynome herzustellen.

Die Menge

$$H_w := \left\{ f \;\middle|\; |f(x) - f(y)| \leq \omega(|x-y|) \;\forall\; x, y \;\in [a,b] \right\}$$

beinhaltet unter anderem die Menge aller auf $[a,b]$ stetigen Funktionen f $\Big($mit $w(t) := \sup\left\{|f(x) - f(y)| : |x-y| \leq t\right\}\Big)$.

Satz 4.3 — (Lebed). Die beste Quadraturformel zu äquidistanten Stützstellen $x_i = a + i \cdot (b-a)/n$ $(i = 0, \ldots, n)$ und Funktionen aus H_ω hat Gewichte $\omega_0 = 1/(2n)$, $\omega_i = 1/n$ $(i = 1, \ldots, n-1)$ und $\omega_n = 1/(2n)$. Des Weiteren gilt für die Genauigkeit

$$\varrho_n(H_w) = 2n \int_0^{\frac{b-a}{2n}} w(t)\, dt, \quad n \in \mathbb{N} \tag{4.9}$$

Nun haben wir deutlich weniger als Differenzierbarkeit vorausgesetzt, aber dies zahlt mit mit einer sehr großen Anzahl an Quadraturpunkten, welches das folgende Beispiel eindrucksvoll darstellt.

MAPLE-Beispiel:

Möchten wir das Integral

$$\int_0^{e^{-2}} \frac{dx}{\log|x|}$$

mit einem möglichst kleinen Fehler, z.B. $< \frac{1}{2}10^{-4}$ und einer summierten Trapezformel berechnen, so liefert der Satz 4.3, dass dazu die Unterteilungsfeinheit von $n = 10^{500}$ nicht ausreicht! Man beachte, dass hier $w(t) = \frac{-1}{\log|t|}$ gilt.

```
> restart:
> n:=10^500:
> 2*n*int(-1/log(x),x=0..exp(-2)/(2*n)
   ):
> evalf(%);

                0.0001171749067
```

Ein weiteres klassisches Ergebnis zur summierten Trapezformel liefert der folgende Satz.

Satz 4.4 — (Kiefer). In der Klasse

$$\mathcal{C}_{A,B} := \{f \text{ wachsend}, f(a) = A, f(b) = B\}$$

ist die summierte Trapezformel zu äquidistanten Stützstellen optimal und es gilt darüber hinaus:

$$\varrho_n(\mathcal{C}_{A,B}) = \frac{(B-A)(b-a)}{2(n+1)}, \quad n \in \mathbb{N}$$

Bemerkung 4.5 Die Fehlerabschätzungen für den Quadraturfehler, die wir im Folgenden betrachten werden, werden eher vom Typ (4.2) sein, d.h. bei dem die Approximationseigenschaft von Funktionen durch Polynome mittels höherer Ableitungen abgeschätzt wird. Die beiden letztgenannten Sätze sollen aber hervorheben, dass man auch unter anderen Voraussetzungen ggf. Konvergenzaussagen bzw. Optimalitätsaussagen treffen kann.

4.2. *Newton-Cotes*-Formeln

Die Kernidee der Mittelpunkts- bzw. der Trapezformel liegt darin, die Funktion f durch eine Interpolierende $P(f)$ zu ersetzen, sodass sich für diese die Quadratur einfach ausführen läßt. Wir verwenden dann $I\big(P(f)\big)$ als Approximation zu $I(f) = \int_a^b f(x)dx$, setzen also:

$$Q[f] = I\big(P(f)\big).$$

Lemma 4.6 Zu $(n+1)$ paarweise verschiedenen Knoten $x_0, \ldots, x_n$ gibt es genau eine Quadraturformel

$$Q[f] = (b-a)\sum_{i=0}^{n} \omega_i f(x_i), \qquad (\omega \widehat{=} \text{ weights}) \tag{4.10}$$

die für alle $P \in \mathbb{P}_n$ vom Grad kleiner oder gleich n exakt ist.

Beweis. Wir verwenden die *Lagrange*-Polynome

$$L_i(x) := \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{(x-x_j)}{(x_i-x_j)},$$

setzen diese in die Quadraturformel ein und erhalten:

$$I\Big(\sum_{i=1}^{n} f(x_i)L_i(x)\Big) = \sum_{i=1}^{n} f(x_i) I\big(L_i(x)\big) = (b-a)\sum_{i=1}^{n} \omega_i f(x_i). \tag{4.11}$$

Dadurch erhalten wir die Gewichte

$$\omega_i = \frac{1}{(b-a)} \int_a^b L_i(x)dx$$

auf eindeutige Weise zurück. ■

Bemerkung 4.7 Die Gewichte ω_j einer Interpolationsquadratur Q können alternativ auch durch Lösen eines linearen Gleichungssystems berechnet werden. Es sei $p_0, \ldots, p_n$ eine Basis von $\mathbb{P}_n$. Es gilt

$$\int_a^b p_j \, dx = \sum_{k=0}^{n} p_j(x_k)\omega_k \quad \text{für alle } j = 0, \ldots, n$$

Diese Gleichung lässt sich als lineares Gleichungssystem formulieren:

$$\begin{pmatrix} p_0(x_0) & \cdots & p_0(x_n) \\ \vdots & & \vdots \\ p_n(x_0) & \cdots & p_n(x_n) \end{pmatrix} \begin{pmatrix} \omega_0 \\ \vdots \\ \omega_n \end{pmatrix} = \begin{pmatrix} \int_a^b p_0 \, dx \\ \vdots \\ \int_a^b p_n \, dx \end{pmatrix} \tag{4.12}$$

Die Matrix auf der linken Seite von (4.13) ist eine transponierte Vandermonde-Matrix und deshalb regulär. Das Gleichungssystem hat als eindeutige Lo?sung den Vektor $(\omega_0, \ldots, \omega_n) \in \mathbb{R}^{n+1}$

MATLAB-Beispiel:

Wir betrachten das kompakte Intervall $[0,1]$ mit der Monombasis. Das Gleichungssystem (4.13) lautet nun

$$\begin{pmatrix} 1 & \cdots & 1 \\ x_0 & \cdots & x_n \\ \vdots & & \vdots \\ x_0^n & \cdots & x_n^n \end{pmatrix} \begin{pmatrix} \omega_0 \\ \vdots \\ \vdots \\ \omega_n \end{pmatrix} = \begin{pmatrix} 1 \\ 1/2 \\ \vdots \\ 1/(n+1) \end{pmatrix} \tag{4.13}$$

und für $x_j = j/n$ erhalten wir die Gewichte der zugehörigen abgeschlossenen Newton-Cotes-Gewichte. Die folgende Matlab-Funktion berechnet die Stützstellen x_j und die zugehörigen Gewichte ω_j.

```
function [x,omega] = ClosedNewtonCotes(n)
x = [0:n]/n;
A = (ones(n+1,1)*x).^([0:n]'*ones(1,n+1));
b = 1./[1:n+1]';
omega = A\b;
% kompakte Version:  omega = (1./[1:n+1])/fliplr(vander([0:n]/n))
```

Für $n \geq 8$ treten auch negative Gewichte auf. Der **Satz von Kusmin** besagt sogar, dass gilt

$$\lim_{n\to\infty} \sum_{j=0}^{n} |\omega_j^{(n)}| = \infty$$

nach dem folgenden Satz 4.10 bekommen wir also im Allgemeinen keine Konvergenz der Quadratur $Q_n[f]$ gegen das Integral $I(f)$.

Definition 4.8 — Newton-Cotes-Formeln. Bei äquidistanter Knotenwahl $a \leq x_0 < x_1 \ldots < x_n \leq b$ heißen die resultierenden Integrationsformeln (4.11) **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

$$x_i = a + \Big(i+1\Big)h \quad \text{mit} \quad h = \frac{b-a}{n+2} \quad (i = 0, \ldots, n) \tag{4.14}$$

gewählt wird. Die Interpolationsquadraturen zu

$$x_i = a + \left(i + \frac{1}{2}\right)h \quad \text{mit} \quad h = \frac{b-a}{n+1} \quad (i = 0, \ldots, n) \tag{4.15}$$

werden als **MacLaurin-Formeln** bezeichnet.

Der Ausdruck für die abgeschlossenen Newton-Cotes-Gewichte ω_i vereinfacht sich durch die Substitution $s := \frac{(x-a)}{h}$ zu

$$\omega_i = \frac{1}{b-a}\int_a^b \prod_{\substack{j=0\\ j\neq i}}^{n} \frac{(x-x_j)}{(x_i-x_j)}\,dx = \frac{1}{n}\int_0^n \prod_{\substack{j=0\\ j\neq i}}^{n} \frac{(s-j)}{(i-j)}\,ds,$$

und für die offenenNewton-Cotes-Formeln erhält man unter der Wahl (4.15) der x_i:

$$\omega_i = \frac{1}{b-a}\int_a^b \prod_{\substack{j=0\\ j\neq i}}^{n} \frac{(x-x_j)}{(x_i-x_j)}\,dx = \frac{1}{n+2}\int_{-1}^{n+1} \prod_{\substack{j=0\\ j\neq i}}^{n} \frac{(s-j)}{(i-j)}\,ds$$

Die in Definition 4.8 definierten Gewichte ω_i sind von den Intervallgrenzen a, b unabhängig und müssen nur einmal zu gegebenem n bestimmt werden.

Einige dieser abgeschlossenen Newton-Cotes-Formeln haben spezielle Namen:

- **Trapezregel** für $n = 1$,
- **Simpson-Regel** oder **Keplersche**[2]-Fassregel für $n = 2$,
- **Newtonsche 3/8-Regel** für $n = 3$,
- **Milne-Regel** für $n = 4$.

Mittels der folgenden Maple-Anweisungen lassen sich die Gewichte einfach bestimmen.

MAPLE-Beispiel: Berechnung der Quadraturgewichte

```
1  > # geschlossene Newton-Cotes-Formel
2  > n:=2:
3  > zaehler:= (x,i) -> quo(product((x-k),k=0..n),(x-i),x):
4  > seq(int(zaehler(x,m)/subs(x=m, zaehler(x,m)),x=0..n)/n,m=0..n);
5
6            1   2   1
7            - , - , -
8            6   3   6
9
10 # offene Newton-Cotes-Formel
11 > n:=2:
12 > zaehler:= (x,i) -> quo(product((x-k),k=0..n),(x-i),x):
13 > seq(int(zaehler(x,m)/subs(x=m, zaehler(x,m)),
14       x=-1..n+1)/(n+2),m=0..n);
```

[2] Kepler, Johannes (1571-1630)

```
15
16          2   -1   2
17          - ,  - ,  -
18          3    3   3
19
20 # MacLaurin-Formel
21 > n:=2:
22 > zaehler:= (x,i) -> quo(product((x-k),k=0..n),(x-i),x):
23 > seq(int(zaehler(x,m)/subs(x=m, zaehler(x,m)),
24       x=-1/2..n+1/2)/(n+1),m=0..n);
25
26          3   1   3
27          - , - , -
28          8   4   8
```

Definition 4.9 — Konvergenz einer Quadraturformel. Wir betrachten im weiteren Verlauf nun Folgen von Quadraturformeln und hierbei den Grenzwert

$$Q_n(f) \xrightarrow{n \to \infty} \int_a^b f(x)dx \quad \text{für jedes } f \in C[a,b]. \tag{4.16}$$

Gilt (4.16), so spricht man von der **Konvergenz der Quadraturformeln** $Q_n (n \in \mathbb{N})$.

Satz 4.10 — *Szegó.* Sei $a \leq x_0^{(n)} < \ldots < x_n^{(n)} \leq b$, und seien $\omega_k^{(n)} \in \mathbb{R}$ $(k = 0, \ldots, n)$. Dann sind die für $n \in \mathbb{N}$ gemäß

$$\begin{aligned} Q_n : \big(C[a,b], \|\cdot\|_\infty\big) &\to \big(\mathbb{R}, |\cdot|\big) \\ f &\mapsto Q_n(f) := \sum_{k=0}^{n} \omega_k^{(n)} f\big(x_k^{(n)}\big) \end{aligned}$$

definierten Quadraturformeln Q_n genau dann konvergent, wenn die beiden folgenden Bedingungen erfüllt sind:

(i)

$$\sup_{n \in \mathbb{N}} \sum_{k=0}^{n} \big|\omega_k^{(n)}\big| < \infty$$

(ii)

$$I(p) = \lim_{n \to \infty} Q_n(p), \qquad \big(p \in \mathbb{P} = \mathbb{P}(a,b)\big)$$

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]. ■

■ **Beispiel 4.11** Wir wenden den Satz von *Szegó* auf die summierte Mittelpunktsregel an:

$$x_k^{(n)} = a + \frac{2k+1}{2} \cdot \frac{b-a}{n+1}, \quad \big(k = 0, \ldots, n\big)$$

mit den Gewichten

$$\omega_k^{(n)} = \frac{b-a}{n+1}.$$

Es gilt: $\sum_k \left|\omega_k^{(n)}\right| = b - a$, demnach ist (i) in Satz 3 erfüllt. Weiterhin gilt (ii) aufgrund der Stetigkeit (damit der *Riemann*-Integrierbarkeit) der Polynome auf $[a, b]$ und der Fehlerabschätzung für die summierte Mittelpunktsregel. ■

Satz 4.12 — *Steklov*. Unter den Voraussetzungen des Satzes von *Szegó* gelte

$$\omega_k^{(n)} \geq 0 \quad \big(n \in \mathbb{N}, 0 \leq k \leq n\big) \tag{4.17}$$

für die Gewichte $\omega_k^{(n)}$ der Quadraturformel Q_n. Dann konvergieren die Quadraturformeln Q_n $(n \in \mathbb{N})$ genau dann, wenn sie für alle $p \in \mathbb{P}$ konvergieren.

Beweis. Nach dem Satz von *Szegó* ist der Beweis erbracht, wenn dort unter der Voraussetzung von (4.17) (i) aus (ii) folgt. Es gelte also (ii) im Satz von *Szegó*. Für $f \equiv 1$ gilt mit (ii)

$$b - a = I(f) = \lim_{n\to\infty} Q_n(f) = \lim_{n\to\infty} \sum_{k=0}^{n} \omega_k^{(n)},$$

was wegen (4.17) die Aussage (i) impliziert. ■

Bemerkung 4.13 Bei den *Newton-Cotes*-Formeln treten durchaus negative Gewichte auf, d.h. der Satz von *Steklov* kann dann nicht angewandt werden. Jedoch hilft auch der Satz 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.

Die folgenden Zeilen *Maple*-Code berechnen die Gewichte der abgeschlossenen *Newton-Cotes*-Formeln für $n = 8$ und die der *MacLaurin*-Formeln für $n = 6$:

Wie man sieht, enthalten beide Quadratur-Formeln negative Gewichte. Für die offenen Newton-Cotes-Formeln erhalten wir schon für $n = 2$ negative Gewichte, wie wir im Maple-Beispiel auf Seite 103 gesehen haben.

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
```

Definition 3 Eine Quadraturformel Q_n hat einen maximalen Exaktheitsgrad $k \in \mathbb{N}$ (oder heißt von der Ordnung $k+1$), falls Polynome vom Grad k noch exakt integriert werden können, d.h.

$$I(q) - Q_n[q] = 0 \qquad \left(q \in \mathbb{P}_k\right),$$

und dabei k maximal gewählt ist.

Satz 4.14 Sind $a \leq x_0^{(n)} < \ldots < x_n^{(n)} \leq b$ und ist Q_n eine Quadraturformel, so gilt für ihre Ordnung k:

$$k \leq 2n+2.$$

Beweis. Die Anwendung der Quadraturformel in der Form

$$Q_n[f] = \sum_{k=0}^{n} \omega_k^{(n)} f\left(x_k^{(n)}\right)$$

auf das Polynom

$$p(x) = \prod_{k=0}^{n} \left(x - x_k^{(n)}\right)^2, \qquad x \in [a,b]$$

vom Grad $2n+2$ liefert die Aussage

$$\int_a^b \underbrace{p(x)}_{>\,0}\, dx - \sum_{k=0}^{n} \omega_k^{(n)} \underbrace{p\left(x_k^{(n)}\right)}_{=\,0} > 0$$

und damit die Behauptung. ∎

4.3. Schwierigkeiten bei der Quadratur

4.3.1. 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*[2]. 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.

4.3.2. Singuläre Integrale

Definition 4.15 — 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 4.16** Zum Beispiel ist die Funktion $1/\sqrt{x}$ auf $[0, 1]$ unbeschränkt und trotzdem besitzt 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 $s = \sqrt{x} \cdot x$:

$$\begin{aligned} \int_0^1 \sqrt{x}\sin(x)\, dx &= \int_0^1 \sqrt{x}\sin(x) - \sqrt{x}\cdot x\, dx + \int_0^1 \sqrt{x}\cdot x\, dx \\ &= \int_0^1 \sqrt{x}\big(\sin(x) - x\big)\, dx + \frac{2}{5} \end{aligned}$$

[2] Bessel, Friedrich Wilhelm (1784-1846)

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

$$\int_0^1 \sqrt{x}\sin(x)\ dx = \int_0^\varepsilon \sqrt{x}\sin(x)\ dx + \int_\varepsilon^1 \sqrt{x}\sin(x)\ dx \qquad 0 < \varepsilon 1$$

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 dar, 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 *Gaußquadratur*
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 in Kapitel (*Gauß*-Gewichte) lassen sich

$$m_k = \int_0^1 \sqrt{x}x^k\ dx = \frac{2}{2k+3}$$

Quadraturpunkte und Gewichte bestimmen.

4.4. Adaptive Quadratur

Motivation Für die Berechnung eines Integrals der Funktion

$$f(x) = \sqrt{x}$$

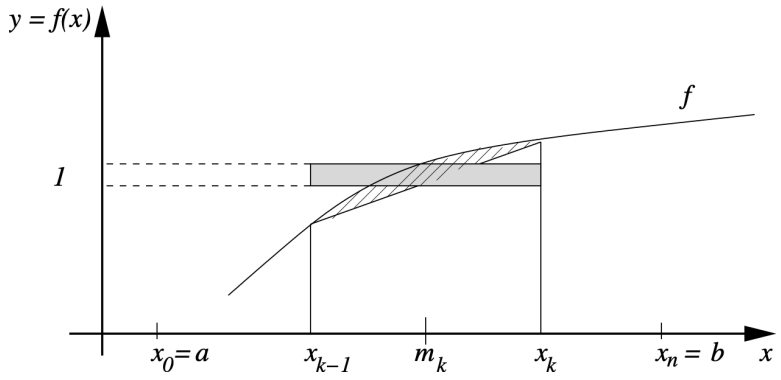
scheint die bisher vorgenommene äquidistante Unterteilung von $[a, b]$ nicht optimal zu sein.

Wir betrachten deshalb die sog. adaptive Quadratur:

Herleitung des Algorithmus Es sei $\sum^n := \{x_j\}_{j=0,\ldots,n}$ mit $a < x_0 < x_1 < \ldots x_n = b$.
Für $f \in \mathcal{C}_{A,B}$ gilt

$$R^k_{k-1}(f) := \int_{x_{k-1}}^{x_k} f(x)dx - \frac{x_k - x_{k-1}}{2}\big(f(x_{k-1}) + f(x_k)\big) \leq (x_k - x_{k-1})|\delta_k|,$$

mit $\delta_k := f(m_k) - \frac{f(x_{k-1})+f(x_k)}{2}$.
Die geometrische Veranschaulichung des Faktors δ_k:



Vorgehensweise Falls die Fehler gleichverteilt sind, d.h. falls gilt:

$$R^k_{k-1}[f] \leq \varepsilon \frac{(x_{k-1} - x_k)}{(b-a)},$$

dann folgt

$$\Big| \int_a^b f(x)dx - Q^{adap.}_{\sum^n}[f] \Big| \leq \sum_{k=1}^{n} R^k_{k-1}[f] \leq \varepsilon.$$

Somit reicht es für jedes Teilintervall

$$|\delta_k| \leq \frac{\varepsilon}{b-a}$$

zu fordern.

In Pseudocodeschreibweise ergibt sich also folgender Algorithmus für die adaptive Quadratur:

Algorithmus 4.4.1: Adaptive Quadratur

Initialisierung: Gegeben $a = x_0 < x_1 < \ldots < x_n = b$

1. Berechne $Q_{k-1}^k[f]$ $(k = 1, \ldots, n)$
2. Falls $\sum(x_k - x_{k-1})|\delta_k| < \varepsilon$ $(\Rightarrow R(f) < \varepsilon)$

$$Q[f] = \sum Q_{k-1}^k[f]$$

sonst

verfeinere $\sum^n \to \sum^{\widetilde{n}}$, d.h. für $k = 1, \ldots, n$

füge $\widetilde{x}_k$ ein, mit $x_{k-1} < \widetilde{x}_k < x_k$ (z.B. durch $\widetilde{x} = \frac{x_{k-1}-x_k}{2}$),

falls $|\delta_k| > \frac{\varepsilon}{b-a}$.

Setze $\widetilde{n} = n$ und gehe zu 1.

Ergebnis:

Die wesentlichen Bestandteile sind also:

- (i) Das Verfeinerungskriterium $|\delta_k| > \frac{\varepsilon}{b-a}$
- (ii) Die Verfeinerungsstrategie, i.e. das Einfügen von $\widetilde{x} = \frac{x_{k-1}-x_k}{2}$
- (iii) Das Abbruchkriterium

4.5. Extrapolation

Motivation Wir betrachten ein Quadraturformel $Q_h[f]$ mit $h = \frac{b-a}{n}$, die eine Entwicklung des Quadraturfehlers in der Form

$$\int_a^b f(x)dx - Q_h[f] = \alpha h^s + \beta h^t + \mathcal{O}\big(h^{t+1}\big), \quad s < t$$

gestattet, wobei α, β von f abhängen aber nicht von h. Dann gilt mit $0 < q < 1$

$$\int_a^b f(x)dx - Q_{qh}[f] = \alpha(qh)^s + \beta(qh)^t + \mathcal{O}\big(h^{t+1}\big)$$

also

$$\int_a^b f(x)dx - \frac{Q_{qh}[f] - q^s Q_h[f]}{1 - q^s} = -\frac{\beta q^s\big(1 - q^{t-s}\big)}{1 - q^s} h^t + \mathcal{O}\big(h^{t+1}\big).$$

In der Linearkombination

$$Q_h^1[f] := \frac{Q_{qh}[f] - q^s Q_h[f]}{1 - q^s}$$

haben wir damit eine Quadraturformel, die sich bezüglich des Quadraturfehlers in h wesentlich günstiger verhält als $Q_h[f]$.

Beispiel 4 Für die summierte Trapezformel gilt die *Euler-Maclaurin*[3]-Summenformel in der Form

$$Q_h^{Trap}[f] = \int_a^b f(x)dx + \sum_{k=1}^{N} c_k h^{2k} + R_{N+1}(h)$$

mit bestimmten von h unabhängigen Koeffizienten c_k und einem Restglied $R_{N+1}(h) = \mathcal{O}\big(h^{2N+2}\big)$ für jedes feste N und $h \to 0$

4.6. 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

$$I[f] = \int_0^1 f(x)e^{i\omega g(x)}dx \tag{4.18}$$

Bemerkung 4 Man beachte, dass sich ein beliebiges Integral $\int_a^b \widetilde{f}(y)e^{i\omega g(y)}dy$ mit der Transformation $y = a + (b-a)x$ auf die Form von (4.18) 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 5 Das übliche numerische Verfahren ein Integral der Form (4.18) 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.

■ **Beispiel 4.17** ■

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 $P[f](x)$ und setzen

$$Q^{Filon}[f] = \int_0^1 P[f](x)e^{i\omega g(x)}dx = \sum_{k=0}^{n} \lambda_k(\omega) f(x_k),$$

[3]Euler, Leonhard (1707-1783)

wobei $\lambda_k(\omega) := \int_0^1 L_k(x)e^{i\omega g(x)}dx$ und $L_k(x)$ sei das *Lagrange*-Polynom zu den Knoten $x_0, \ldots, x_n$. Man beachte dabei, dass die Konstruktion von Q^{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

$$Q^{Filon}[f] - I[f] \sim \begin{cases} \mathcal{O}(\omega^{-1}), & , \quad x_0 > 0 \text{ oder } x_n < 1 \\ \mathcal{O}(\omega^{-2}), & , \quad x_0 = 0 \text{ und } x_n = 1 \end{cases} \quad \omega \to \infty$$

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 5 Wie oben sei $f(x) = \cos(x), g(x) = x$ und $\omega > 0$. Das Polynom

$$p(x) = \sum_{k=1}^{n} f(x_k) \cdot \Big(1 - \frac{\omega''(x_n)}{\omega'(x_k)}(x - x_k)\Big) L_k^2(x) = \sum_{k=1}^{n} f'(x_k) \cdot (x - x_k) L_k^2(x),$$

wobei L_k die *Lagrange*-Polynome sind und $\omega(x) = (x - x_1) \cdot \ldots \cdot (x - x_n)$ ist, erfüllt das *Hermite'sche* Interpolationsproblem

$$p(x_k) = f(x_k), \quad p'(x_k) = f'(x_k) \qquad (k = 1, 2, \ldots, n);$$

für $n = 2$ lautet

$$\begin{aligned} p(x) \quad = \quad & \frac{(3x_1 - x_2 - 2x)(x - x_2)^2}{(x_1 - x_2)^2} f(x_1) + \frac{(3x_2 - x_1 - 2x)(x - x_1)^2}{(x_2 - x_1)^2} f(x_2) \\ & + \frac{(x - x_1)(x - x_2)^2}{(x_1 - x_2)^2} f'(x_1) + \frac{(x - x_1)^2(x - x_2)}{(x_2 - x_1)^2} f'(x_2). \end{aligned}$$

Setzen wir $x_1 = 0$ und $x_2 = 1$, 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

```
1 > collect(int((1+2*x)*(x-1)^2*exp(I*omega*x) ,x=0..1), omega);
2   collect(int((3-2*x)*x^2    *exp(I*omega*x) ,x=0..1), omega);
3   collect(int(x*(x-1)^2      *exp(I*omega*x) ,x=0..1), omega);
4   collect(int(x^2*(x-1)      *exp(I*omega*x) ,x=0..1), omega);
```

Die entsprechende Quadraturformel lautet dann

$$\begin{aligned} Q^{Filon}[f] &= \Big(\frac{i}{\omega} + \frac{6i(1+e^{i\omega})}{\omega^3} + \frac{12(1-e^{i\omega})}{\omega^4}\Big) f(0) \\ &+ \Big(\frac{-ie^{i\omega}}{\omega} - \frac{6(1+e^{i\omega})}{\omega^3} - \frac{12(1-e^{i\omega})}{\omega^4}\Big) f(1) \\ &+ \Big(-\frac{1}{\omega^2} + \frac{2i(2+e^{i\omega})}{\omega^3} + \frac{6(1-e^{i\omega})}{\omega^4}\Big) f'(0) \\ &+ \Big(\frac{e^{i\omega}}{\omega^2} + \frac{2i(1+2e^{i\omega})}{\omega^3} + \frac{6(1-e^{i\omega})}{\omega^4}\Big) f'(1) \end{aligned}$$

Für $f(x) = \cos(x), g(x) = x$ und $10 < \omega < 100$ ist der Fehler für Q^{Filon} skaliert mit ω^3 in Abbildung 4.3 dargestellt.

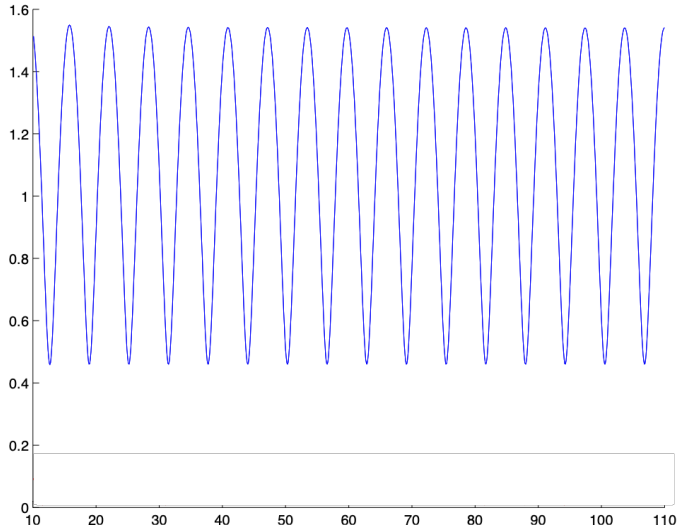


Abbildung 4.3.: Der Fehler von Q^{Filon}, skaliert mit ω^3, für $f(x) = \cos(x), g(x) = x$ und $10 < \omega < 100$.

5. Orthogonalpolynome und Gauß-Quadratur

Motivation Bei vielen Problemen der Mathematik lässt sich eine Lösung bzw. eine Lösungsfunktion bezüglich sog. speziellen Funktionen entwickeln, die sich dann durch ganz spezielle, dem Problem angemessene Eigenschaften auszeichnen. Beispiele hierfür sind etwa

(*i*) *Tschebyscheff*[1]-Polynome (siehe Kapitel über Interpolation),
(*ii*) *Legendre*[2]-Polynome (siehe Kapitel über Gauß-Quadratur).

Diese Funktionen zeichnen sich insbesondere durch ihre Orthogonalität aus.

5.1. Allgemeine Charakteristika

Definition 5.1 — 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 α, $\beta \in \mathbb{R}$ 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

$$(\alpha x + \beta y, z) = \alpha(x,z) + \beta(y,z)\,.$$

Satz 5.2 — Cauchy-Schwarz-Ungleichung. Es sei V ein reeller Vektorraum, $(\cdot,\cdot)\colon V \times V \to \mathbb{R}$ ein **Skalarprodukt** auf V. Dann gilt

$$(x,y) \leq \sqrt{(x,x)}\sqrt{(y,y)}\,.$$

[1] Tschebyscheff, Pafnuti Lwowitsch (1821-1894), auch als Čebyšv, Chebyshev, Tschebyschow, Tschebyschew oder Tschebyschev transkribiert
[2] Legendre, Adrien-Marie (1752-1833)

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. ■

Satz 5.3 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 5.4 — 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 5.5 Im Folgenden setzen wir voraus, dass die durch das Skalarprodukt induzierte Norm

$$\| q \| := \| q \|_\omega := \sqrt{(q, q)_\omega}$$

für alle Polynome $q \in \mathbb{P}_k$ $(k \in \mathbb{N})$ wohldefiniert und endlich ist. Somit existieren auch die **Momente**

$$m_k := \int_a^b \omega(x) x^k\, dx,$$

da mit der **Cauchy-Schwarzsch-Ungleichung** folgt:

$$|m_k| = |(1, x^k)_\omega| \leq \|1\|_\omega \|x^k\|_\omega < \infty.$$

■ **Beispiel 5.6** Einige Beispiele von Gewichtsfunktionen seien hier genannt, die ungewöhnlich genug sind, um numerische Techniken zu erfordern, und in praktischen Anwendungen verwendet werden.

- $\omega(x) = x^\alpha \log(1/x)$ auf $[0,1]$ mit $\alpha > 0$.
  Die Momente $m_k = (k+\alpha+1)^{-2}$ 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 $[0,c]$ $(0 < c < \infty)$.
  Dies sind Laguerre bzw. Hermite-Gewichte auf einem endlichen Intervall. Die Momente m_k lassen sich durch die unvollständige Gamma-Funktion $\gamma(\alpha, x) = \int_0^x t^{\alpha-1} e - t\, dt$ ausdrücken, nämlich $m_k = \gamma(k+1, c)$ bzw. $m_k = \frac{1}{2}\gamma(\frac{1}{2}(k-1), c^2)$. Beide Varianten finden Anwendung bei der Gauss-Quadratur von Integralen in der molekularen Quantenmechanik.

■

Definition 5.7 — Orthogonalpolynome. Es sei (p_k) eine Folge paarweise orthogonaler Polynome $p_k \in \mathbb{P}_k$, d.h.

$$(p_i, p_j)_\omega = \delta_{ij}(p_i, p_i)_\omega \geq 0 \tag{5.1}$$

und exakt vom Grad k, dann heißen die p_k **Orthogonalpolynome** über $[a,b]$ bzgl. der Gewichtsfunktion ω. Die Normalisierung $\tilde{p}_k = p_k / \sqrt{(p_k, p_k)_\omega}$ $(k = 0, 1, 2, \ldots)$ liefert die **orthonormalen Polynome**, indexPolynome!orthonormalefür die gilt

$$(\tilde{p}_i, \tilde{p}_j)_\omega = \delta_{ij} := \begin{cases} 0 & \text{, falls } i \neq j \\ 1 & \text{, sonst.} \end{cases} \tag{5.2}$$

Aufgabe 5.8 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 5.9

1. Da die $p_0, \ldots, p_k$ eine Basis des $\mathbb{P}_k$ darstellen, folgt sofort

$$(p_k, q)_\omega = (p_k, \sum_{i=0}^{k-1} \alpha_i p_i)_\omega = 0 \quad \text{für alle } q \in \mathbb{P}_{k-1}\,. \tag{5.3}$$

2. Die Definition der Orthogonalpolynome über (5.1) ist keineswegs eindeutig. Eine mögliche Standardisierung ist z.B.

$$p_k(x) = x^k + a_{k-1}x^{k-1} + \ldots\,,$$

d.h. der führende Koeffizient ist Eins.

Definition 5.10 — Monisches Polynom. Ist der führende Koeffizient eines Polynoms Eins, so bezeichnet man dieses als **monisch**.

Satz 5.11 Zu jedem positiv gewichteten Skalarprodukt $(\cdot,\cdot)_\omega$ gibt es eindeutig bestimmte monische Orthogonalpolynome $p_k \in \mathbb{P}_k$ (d.h. mit führendem Koeffizienten Eins). Gilt zusätzlich $(x\,p_k, p_j)_\omega = (p_k, x\,p_j)_\omega$ für $j,k \geq 0$, dann erfüllen die Orthogonalpolynome die folgende Drei-Term-Rekursion:

$$p_k(x) = (x-\beta_k)p_{k-1}(x) - \gamma_k p_{k-2}(x), \quad k = 1,2,\ldots \tag{5.4}$$

mit $p_{-1} := 0$, $p_0 := 1$ und

$$\beta_k = \frac{(xp_{k-1}, p_{k-1})_\omega}{(p_{k-1}, p_{k-1})_\omega}, \quad \gamma_k = \frac{(p_{k-1}, p_{k-1})_\omega}{(p_{k-2}, p_{k-2})_\omega}. \tag{5.5}$$

Beweis von Satz 5.11. Man sieht unmittelbar, dass $p_0 \equiv 1 \in \mathbb{P}_0$ gilt, den Rest der Behauptung zeigen wir induktiv. Seien also $p_0,\ldots,p_{k-1}$ paarweise orthogonale Polynome mit $p_j \in \mathbb{P}_j$ vom Grad j und führendem Koeffizienten gleich Eins. Ist $p_k \in \mathbb{P}_k$ ebenso normiert, dann folgt

$$p_k(x) - xp_{k-1}(x) \quad \text{ist vom Grade } \leq k-1.$$

Da jedoch $p_0,\ldots,p_{k-1}$ eine Orthogonalbasis von $\mathbb{P}_{k-1}$ bzgl. $(\cdot,\cdot)_\omega$ bilden, gilt

$$xp_{k-1} - p_k = \sum_{j=0}^{k-1} c_j p_j \quad \text{mit} \quad c_j := \frac{(xp_{k-1} - p_k, p_j)_\omega}{(p_j, p_j)_\omega}.$$

(Man setze die linke Gleichung in $(\cdot, p_\ell)_\omega$ für $\ell = 0,\ldots,k-1$ ein.) Außerdem folgt aus der Orthogonalität von p_k zu $p_0,\ldots,p_{k-1}$

$$c_j = \frac{(xp_{k-1}, p_j)_\omega}{(p_j,p_j)_\omega} = \frac{(p_{k-1}, xp_j)_\omega}{(p_j,p_j)_\omega}\,.$$

Da das Produkt xp_j als Linearkombination von $p_0,\ldots,p_{j+1} \in \mathbb{P}_{j+1}$ geschrieben werden kann, gilt somit $c_j = 0$ für $j+1 < k-1$, d.h.

$$c_0 = \ldots = c_{k-3} = 0 \quad \text{bzw. es folgt} \quad xp_{k-1} - p_k = c_{k-2}p_{k-2} + c_{k-1}p_{k-1}\,.$$

Beachtet man $xp_{k-2} = p_{k-1} + q$ mit $q \in \mathbb{P}_{k-2}$, so folgt

$$c_{k-2} = \frac{(p_{k-1}, xp_{k-2})_\omega}{(p_{k-2},p_{k-2})_\omega} = \frac{(p_{k-1},p_{k-1})_\omega}{(p_{k-2},p_{k-2})_\omega}$$

und damit für $k = 1,2,\ldots$

$$p_k(x) = \left(x - \frac{(xp_{k-1},p_{k-1})_\omega}{(p_{k-1},p_{k-1})_\omega}\right)p_{k-1}(x) - \frac{(p_{k-1},p_{k-1})_\omega}{(p_{k-2},p_{k-2})_\omega}p_{k-2}(x)\,.$$

■

Bemerkung 5.12 Dass sich nicht alle Orthogonalpolynome durch eine Drei-Term-Rekursion darstellen lassen, soll das folgenden Beispiel illustrieren. Die Abbildung $((f,g)) := -\int_{-1}^{1}\int_{-1}^{1} f(x)g(y)\log|x-y|\,dx\,dy$ ist ein Skalarprodukt auf der Menge der stetigen Funktionen. Mit dem Gram-Schmidtschen-Orthogonalsierungsverfahren erhält man die Orthogonalpolynome beginnend mit $p_0 := 1$ mittels

$$p_{k+1}(x) = xp_k(x) - \sum_{j=0}^{k} \frac{((xp_k, p_j))}{((p_j, p_j))} p_j(x), \quad (k = 1, 2, \ldots)$$

Mit Hilfe von Maple lassen sich die Integrale schnell berechnen und man erhält:

$$\begin{aligned} p_1(x) &= x \\ p_2(x) &= x^2 - \frac{8 - 6\log(2)}{27 - 18\log(2)} \\ p_3(x) &= x^3 - \frac{5}{9}x \\ p_4(x) &= x^4 - \frac{338 - 240\log(2)}{425 - 300\log(2)}x^2 + \frac{29 - 20\log(2)}{425 - 300\log(2)} \end{aligned}$$

mit

$$p_4(x) = xp_3(x) - \frac{917 - 660\log(2)}{3825 - 2700\log(2)}p_2(x) - \frac{17 - 60\log(2)}{6075 - 4050\log(2)}p_0(x)\,.$$

Dies entspricht aber nicht der Darstellung aus drei aufeinanderfolgender Terme wie in (5.4).

Satz 5.13 Es seien $\tilde{p}_k$ $(k = 0, 1, 2\ldots)$ die Orthonormalpolynome bzgl. des Skalarprodukts $(\cdot,\cdot)_\omega$. Dann gilt

$$\begin{aligned} \sqrt{\gamma_{k+1}}\,\tilde{p}_{k+1}(x) &= (x - \beta_k)\,\tilde{p}_k(x) - \sqrt{\gamma_k}\,p_{k-1}(x) \quad k = 0, 1, 2, \ldots & (5.6) \\ p_{-1}(x) &= 0, \quad p_0(x) = 1/\sqrt{\gamma_0}\,, & (5.7) \end{aligned}$$

wobei die β's und γ's durch (5.5) gegeben sind.

Definition 5.14 Wir definieren die unendlich-dimensionale, symmetrische und tridiagonale **Jacobi**-Matrix

$$J_\infty = J_\infty(\omega) := \begin{pmatrix} \beta_0 & \sqrt{\gamma_1} & & & 0 \\ \sqrt{\gamma_1} & \beta_1 & \sqrt{\gamma_2} & & \\ & \sqrt{\gamma_2} & \beta_2 & \sqrt{\gamma_3} & \\ & & \ddots & \ddots & \ddots \\ 0 & & & & \end{pmatrix}.$$

Die führende $n \times n$ Untermatrix wird bezeichent mit

$$J_n = J_n(\omega) = \big(J_\infty(\omega)\big)_{i,j=1}^n$$

Falls man die ersten n Gleichungen in (5.6) in der Form

$$x\,\tilde{p}_k(x) = \sqrt{\gamma_k}\,\tilde{p}_{k-1}(x) + \beta_k \tilde{p}_{k+1}(x) + \sqrt{\gamma_{k+1}}\,\tilde{p}_{k+1}(x) \quad k = 0, 1, 2, \ldots, n-1 \tag{5.8}$$

schreibt und

$$\tilde{\boldsymbol{p}}(x) = (\tilde{p}_0(x), \tilde{p}_1(x), \ldots, \tilde{p}_{n-1}(x))^T$$

definiert, dann lässt sich (5.8) wie folgt schreiben:

$$x\,\tilde{\boldsymbol{p}}(x) = J_n(\omega)\,\tilde{\boldsymbol{p}}(x) + \sqrt{\gamma_n}\,\tilde{p}_n(x)\boldsymbol{e_n} \tag{5.9}$$

mit $\boldsymbol{e_n} = (0, \ldots, 0, 1)^T \in \mathbb{R}^n$.

Satz 5.15 Die Nullstellen $x_i^{(n)}$ $(i = 1, \ldots, n)$ von p_n (oder $\tilde{p}_n$) sind die Nullstellen der $n \times n$-Jacobi-Matrix $J_n(\omega)$ und $\tilde{\boldsymbol{p}}_{\boldsymbol{n}}(x_i^{(n)})$ sind die zugehörigen Eigenvektoren.

Beweis. Die Behauptung folgt direkt aus (5.9), indem man $x = x_i^{(n)}$ einsetzt und beachtet, dass $\tilde{\boldsymbol{p}}(x_i^{(n)}) \neq 0$ gilt, da die erste Komponente von $\tilde{\boldsymbol{p}}(x_i^{(n)})$ gerade $1/\sqrt{\gamma_0}$. ■

Bemerkung 5.16 Allgemeiner ist die Darstellung der Orthogonalpolynome in der Form

$$p_n(x) = k_n x^n + k_n' x^{n-1} + \ldots \qquad (n = 0, 1, 2 \ldots)\,. \tag{5.10}$$

Zusätzlich definieren wir

$$h_n(p_n) := h_n := \int_a^b \omega(x) p_n^2(x)\,dx = \|p_n\|_\omega^2\,. \tag{5.11}$$

Übertragen wir nun die Drei-Term-Rekursion aus Satz 5.11 für Orthogonalpolynome mit führendem Koeffizienten Eins auf die allgemeinere Form (5.10), so erhalten wir folgendes Resultat:

Satz 5.17 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:

$$p_n(x) = (a_n x - b_n) p_{n-1} - c_n p_{n-2} \quad (n = 1, 2, \ldots), \tag{5.12}$$

mit $p_{-1} = 0$, $p_0 = k_0$. Die Koeffizienten ergeben sich wie folgt:

$$a_n = \frac{k_n}{k_{n-1}},\ b_n = a_n\Big(\frac{k'_{n-1}}{k_{n-1}} - \frac{k'_n}{k_n}\Big),\ c_n = \frac{k_n k_{n-2} h_{n-1}}{k_{n-1}^2 h_{n-2}} \quad (n = 2, 3, \ldots)\,. \tag{5.13}$$

Beweis. Die Behauptung folgt aus Satz 5.11 mit der Darstellung aus Bemerkung 5.16. 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 5.11 liefert

$$p_n(x) = k_n\,\bar{p}_n(x) = k_n(x - \beta_n)\bar{p}_{n-1}(x) - k_n\gamma_n\bar{p}_{n-2}(x), \quad n = 1, 2, \ldots \tag{5.14}$$

mit $\bar{p}_{-1} := 0$, $\bar{p}_0 := p_0/k_0 = 1$ und

$$\beta_n = \frac{(x\bar{p}_{n-1}, \bar{p}_{n-1})_\omega}{(\bar{p}_{n-1}, \bar{p}_{n-1})_\omega}$$

sowie

$$\gamma_n = \frac{(\bar{p}_{n-1}, \bar{p}_{n-1})_\omega}{(\bar{p}_{n-2}, \bar{p}_{n-2})_\omega} = \frac{1/k_{n-1}^2\,(p_{n-1}, p_{n-1})_\omega}{1/k_{n-2}^2\,(p_{n-2}, p_{n-2})_\omega} = \frac{k_{n-2}^2}{k_{n-1}^2}\frac{h_{n-1}}{h_{n-2}}\,. \tag{5.15}$$

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.

$$\beta_n = \frac{(x\bar{p}_{n-1}, \bar{p}_{n-1})_\omega}{(\bar{p}_{n-1}, \bar{p}_{n-1})_\omega} = \frac{k'_{n-1}}{k_{n-1}} - \frac{k'_n}{k_n}\,. \tag{5.16}$$

Somit ergibt sich mit (5.15)

$$k_n\gamma_n\bar{p}_{n-2}(x) = k_n\frac{k_{n-2}^2}{k_{n-1}^2}\frac{h_{n-1}}{h_{n-2}}\frac{1}{k_{n-2}}p_{n-2}(x) = \frac{k_n k_{n-2}}{k_{n-1}^2}\frac{h_{n-1}}{h_{n-2}}p_{n-2}(x)$$

bzw.

$$k_n\,x\,\bar{p}_{n-1}(x) = \frac{k_n}{k_{n-1}}\,x\,p_{n-1}(x)$$

und mit (5.16)

$$k_n\beta_n\bar{p}_{n-1}(x) = \frac{k_n}{k_{n-1}}\left(\frac{k'_{n-1}}{k_{n-1}} - \frac{k'_n}{k_n}\right)p_{n-1}(x)\,.$$

Die letzten drei Gleichungen mit (5.1) liefern dann die Behauptung. ■

Bemerkung 5.18 Für Orthogonalpolynome gilt zur Angabe der p_n die sog. *Rodrigues*[a]-Formel; d.h. alle p_n erfüllen die folgende explizite Formel

$$p_n(x) = \frac{1}{e_n\omega(x)}\frac{d^n}{dx^n}\Big\{\omega(x)\big(g(x)\big)^n\Big\}, \quad n \in \mathbb{N}_0 \tag{5.17}$$

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.

[a] Rodrigues, Olinde (1794-1851)

■ **Beispiel 5.19** 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*[3]	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* $\alpha,\beta>-1$	-1	1	$(1-x)^\alpha\cdot$ $\cdot(1+x)^\beta$	$P_n^{(\alpha,\beta)}(1)=\binom{n+\alpha}{n}$		$(-2)^n n!$	$1-x^2$

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 5.20 Mit Hilfe der Rodrigues-Formel zeige man, dass für die Legendre-Polynome gilt

$$\int_{-1}^{1} P_m(x)P_n(x)\,dx = \begin{cases} 0 & \text{falls } m\neq n \\ \frac{2}{2n+1} & \text{falls } m=n\,. \end{cases} \qquad (m,n\in\mathbb{N}_0)$$

■

Satz 5.21 — Christoffel-Darboux. Mit der für Orthogonalpolynome zuvor eingeführten Notation (5.10), (5.11) gilt

$$\sum_{i=0}^{n}\frac{p_i(x)p_i(y)}{h_i} = \frac{k_n}{k_{n+1}\cdot h_n}\cdot\frac{p_{n+1}(x)p_n(y)-p_n(x)p_{n+1}(y)}{x-y}\,. \qquad (5.18)$$

Beweis. Für $n=0$ erhält man die Identität (5.18) durch Einsetzen von $p_0(x)=k_0$ und $p_1(x)=k_1x+k_1'$ und für $n\geq 1$ aus der Rekursionsformel (5.12). Aus dieser folgt nämlich für $n\geq 1$ durch jeweiliges Einsetzen für p_{n+1}

$$\begin{aligned} & p_{n+1}(x)p_n(y)-p_n(x)p_{n+1}(y) \\ &= \big((a_{n+1}+b_{n+1}x)p_n(x)+c_{n+1}\,p_{n-1}(x)\big)p_n(y) \\ &\quad -p_n(x)\big((a_{n+1}+b_{n+1}\,y)p_n(y)+c_{n+1}\,p_{n-1}(y)\big) \\ &= b_{n+1}(x-y)p_n(x)p_n(y)+c_{n+1}\big(p_{n-1}(x)p_n(y)-p_n(x)p_{n-1}(y)\big) \end{aligned}$$

[3] Laguerre, Edmond Nicolas (1834-1886)

Mit (5.13) 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.

$$\begin{aligned} & \frac{k_n}{k_{n+1}} \cdot \frac{p_{n+1}(x)p_n(y) - p_n(x)p_{n+1}(y)}{x-y} \\ = \; & p_n(x)p_n(y) + \frac{k_{n+1}k_{n-1}h_n k_n}{k_n^2 \; h_{n-1} \; k_{n+1}} \cdot \frac{p_n(x)p_{n-1}(y) - p_{n-1}(x)p_n(y)}{x-y} \qquad (5.19) \\ = \; & p_n(x)p_n(y) + \frac{k_{n-1}h_n}{k_n h_{n-1}} \cdot \frac{p_n(x)p_{n-1}(y) - p_{n-1}(x)p_n(y)}{x-y}. \end{aligned}$$

Gelte nun (5.18) für ein $n-1 \geq 0$ dann schließ en wir mit (5.19) 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 5.22 Man beachte den Spezialfall $y \to x$ in Satz 5.21. 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}

$$p'_{n+1}(x^*)\, p_n(x^*) = \frac{k_{n+1}h_n}{k_n} \sum_{i=0}^{n} \frac{p_i^2(x^*)}{h_i}. \qquad (5.20)$$

Satz 5.23 Es sei $\omega \in C(a,b)$, $\omega(x) > 0$ $(a < x < b)$ 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 $x_1, \ldots, x_m$ die $m \leq k$ verschiedenen Nullstellen $a < x_i < b$, an denen p_k sein Vorzeichen wechselt. Das Polynom

$$q(x) := (x - x_1) \cdot \ldots \cdot (x - x_m)$$

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 steht, folgt unmittelbar: grad $q = m \geq k$ und damit die Behauptung. ∎

Satz 5.24 Die Nullstellen von p_{n+1} wechseln sich mit denen von p_n ab, d.h.

$$x_1^{(n+1)} < x_1^{(n)} < x_2^{(n+1)} < x_2^{(n)} < \cdots < x_n^{(n)} < x_{n+1}^{(n+1)},$$

wobei $x_i^{(n+1)}$, $x_i^{(n)}$ die Nullstellen von p_{n+1} bzw. p_n in aufsteigender Reihenfolge seien.

Beweis. Aus 5.20 folgt, dass $p'_{n+1}(x^*)p_n(x^*)$ für alle Nullstellen x^* von p_{n+1} ein festes Vorzeichen besitzt. Seien nun μ und ν zwei aufeinander folgende Nullstellen von p_{n+1}. Da nach Satz 5.23 die Nulllstellen von p_{n+1} einfach sind, gilt $p'_{n+1}(\mu)p'_{n+1}(\nu) < 0$ und somit auch $p_n(\mu)p_n(\nu) < 0$. Dies bedeutet, dass zwischen zwei Nullstellen von p_{n+1} eine Nullstelle von p_n liegt, womit die Behauptung beweisen ist. ∎

Satz 5.25 Es sei $\omega \in C(a,b)$ eine positive Gewichtsfunktion auf (a,b) und (p_k) eine Folge von Orthogonalpolynomen bzgl. des Skalarprodukts $(\cdot,\cdot)_\omega$. Des Weiteren sei $n \in \mathbb{N}$ und $x_1 < \ldots < x_n$ die Nullstellen von $p_n(x)$. Dann existieren (von n abhängige) eindeutig bestimmte Gewichte $\omega_1, \ldots, \omega_n \in \mathbb{R}_+$, sodass gilt:

$$\int_a^b \omega(x)q(x)\,dx = \omega_1\, q(x_1) + \ldots + \omega_n\, q(x_n) \quad \text{für alle } q \in \mathbb{P}_{2n-1}\,. \tag{5.21}$$

Bemerkung 5.26 Diese Art der Integralberechnung wird als Gauß-Quadratur bezeichnet.

Beweis zu Satz 5.25. Es sei p_n das bis auf Vielfachheit eindeutig bestimmte Orthogonalpolynom bzgl. $(\cdot,\cdot)_\omega$ vom echten Grad n. Dann existiert zu jedem $q \in \mathbb{P}_{2n-1}$ die

eindeutige Darstellung $q = \varphi p_n + \psi$ mit $\varphi, \psi \in \mathbb{P}_{n-1}$. Das Ausnutzen der Orthogonalität von $(p_n, x^k)_\omega = 0$ für $k = 0, \ldots, n-1$ liefert

$$\int_a^b \omega(x)q(x)\,dx = \int_a^b \omega(x)\varphi(x)p_n(x)\,dx + \int_a^b \omega(x)\psi(x)\,dx = \int_a^b \omega(x)\psi(x)\,dx. \quad (5.22)$$

Seien $L_1(x), \ldots, L_n(x) \in \mathbb{P}_{n-1}$ die *Lagrange*-Polynome

$$L_k(x) = \prod_{\substack{j=1 \\ j\neq k}}^{n} \frac{x - x_j}{x_k - x_j} \in \mathbb{P}_{n-1} \quad k \in \{1, \ldots, n\}$$

(vgl. hierzu Kapitel 2, Interpolation) zu $x_1, \ldots, x_n$. Man beachte die Eigenschaft $L_k(x_j) = \delta_{jk}$, wobei δ_{jk} das Kronecker-Symbol ist. Es gilt dann folgende Darstellung

$$\psi(x) = \sum_{i=1}^{n} \psi(x_i)L_i(x)$$

und somit für (5.22)

$$\int_a^b \omega(x)q(x)\,dx = \int_a^b \omega(x)\Big(\sum_{i=1}^{n} \psi(x_i)L_i(x)\Big)\,dx = \sum_{i=1}^{n} \psi(x_i) \int_a^b \omega(x)L_i(x)\,dx.$$

Dies ist aber genau die Darstellung (5.21) mit $\omega_i = \int_a^b \omega(x)L_i(x)dx$, da an den Nullstellen von p_n gilt $q(x_i) = \varphi(x_i)\,p_n(x_i) + \psi(x_i) = \psi(x_i)$.
Beweisen wir nun die Eindeutigkeit der ω_i. Dazu seien $\omega_1, \ldots, \omega_n$ und $\widetilde{\omega_1}, \ldots, \widetilde{\omega_n}$ zwei verschiedene Wahlen der Koeffizienten in (5.21). Durch Differenzbildung erhalten wir also

$$0 = \sum_{i=1}^{n} \psi(x_i)(\omega_i - \widetilde{\omega_i}),$$

setzen wir nacheinander $\psi(x) = L_k(x)$ $(k = 1, \ldots, n)$, so folgt sofort $\omega_k = \widetilde{\omega_k}$ für alle $k \in \{1, \ldots, n\}$.

Zuletzt müssen wir noch die Positivität der ω_i nachweisen. Setzen wir dazu $q(x) = L_k^2(x)$, dann folgt mit $\omega(x) > 0$ $\big(x \in (a, b)\big)$ und (5.21)

$$0 < \int_a^b \omega(x)L_k^2(x)\,dx = \sum_{i=1}^{n} \omega_i\, L_k^2(x_i) = \omega_k$$

und damit die Behauptung. ■

Satz 5.27 Für die Gewichte ω_k $(k = 1, \ldots, n)$ in Satz 5.25 gelten die Darstellungen

$$\omega_k = \frac{k_n\, h_{n-1}}{p_n'(x_k)\, p_{n-1}(x_k)\, k_{n-1}} = \left(\sum_{i=0}^{n-1} \frac{p_i^2(x_k)}{h_i}\right)^{-1} \qquad (k = 1, \ldots, n)\,,$$

wobei x_k die Nullstellen von p_n seien.

Beweis. Es sei x_k, $k \in \{1, \dots, n\}$, eine Nullstelle von p_n. Dann gilt

$$\lim_{x \to x_k} \frac{p_n(x)}{x - x_k} = \lim_{x \to x_k} \frac{p_n(x) - p_n(x_k)}{x - x_k} = p_n'(x_k)\,.$$

Wir definieren

$$q(x) = \begin{cases} p_n(x)\, p_{n-1}(x)/(x - x_k) & \text{falls } x \in \mathbb{R} \setminus \{x_k\} \\ p_n'(x_k)\, p_{n-1}(x_k) & \text{falls } x = x_k \end{cases}\,.$$

Da $q \in \mathbb{P}_{2n-2}$, gilt

$$\int_a^b \omega(x) q(x)\, dx = \sum_{j=1}^{n} \omega_j\, q(x_j) = \omega_k\, q(x_k) = \omega_k\, p_n'(x_k)\, p_{n-1}(x_k)\,. \tag{5.23}$$

Nutzen wir die Darstellung $p_n(x) = \frac{k_n}{k_{n-1}}(x - x_k)p_{n-1}(x) + (x - x_k)r(x)$ mit einem $r \in \mathbb{P}_{n-2}$, so ergibt sich

$$\begin{aligned} \int_a^b \omega(x) q(x)\, dx &= \int_a^b \omega(x) \left(\frac{k_n}{k_{n-1}} p_{n-1}(x) + r(x) \right) p_{n-1}(x)\, dx \\ &= \frac{k_n}{k_{n-1}} \int_a^b \omega(x) p_{n-1}^2(x)\, dx = \frac{k_n\, h_{n-1}}{k_{n-1}}\,. \end{aligned} \tag{5.24}$$

Aus (5.23) und (5.24) erhalten wir die erste und daraus mit (5.20) die zweite Darstellung . ∎

Zum Ende des Abschnitts beschäftigen wir uns mit der Frage, wie sich der Fehler zwischen exakter Integration und Quadratur für eine Klasse von Funktionen bestimmen lässt. Vorbereitend benötigen wir jedoch noch den Mittelwertsatz der Integralrechnung.

Satz 5.28 — Mittelwertsatz der Integralrechnung. Es sei $f : [a, b] \to \mathbb{R}$ stetig auf $[a, b]$, g sei Riemann-integrierbar mit $g(x) \geq 0$ (≤ 0) für alle $x \in [a, b]$. Dann gilt

$$\int_a^b f(x)g(x)\ dx = f(\xi) \int_a^b g(x)\ dx \quad \text{für mindestens ein } \xi \in [a, b].$$

Beweis. Nach Voraussetzung gilt $g(x) \geq 0$ für alle $x \in [a, b]$. Gilt $\int_a^b g(x)\, dx = 0$, so ist $g(x) = 0$ bis auf höchstens endlich viele Ausnahmen, da g Riemann-integrierbar und $g(x) \geq 0$ $(x \in [a, b])$. Hieraus ergibt sich $\int_a^b f(x)g(x)\, dx = 0$ und somit die Behauptung.

Gelte nun $\int_a^b g(x)\, dx \neq 0$. Mit der Notation

$$m := \min_{\xi \in [a,b]} f(\xi) \leq f(x) \leq \max_{\xi \in [a,b]} f(\xi) =: M$$

folgt

$$m \int_a^b g(x)\ dx \leq \int_a^b f(x)g(x)\ dx \leq M \int_a^b g(x)\ dx.$$

Mit dem Zwischenwertsatz folgt aufgrund der Stetigkeit von f, dass ein $\xi \in [a,b]$ existiert mit

$$f(\xi) = \frac{\int_a^b f(x)g(x)\,dx}{\int_a^b g(x)\,dx},$$

womit die Aussage bewiesen wäre. ■

Satz 5.29 — A. Markoff. Für jede Funktion $f \in C^{2n}[a,b]$ lässt sich der Quadraturfehler der *Gauß* -Quadratur $Q_n[f] = \sum_{i=1}^n w_i f(x_i)$ ausdrücken durch

$$\int_a^b \omega(x)f(x)\,dx - \sum_{i=1}^n w_i f(x_i) = \frac{f^{(2n)}(\xi)}{(2n)!} \cdot \frac{(p_n,p_n)_\omega}{k_n^2},$$

für ein $\xi \in [a,b]$, wobei k_n der führende Koeffizient des n-ten Orthogonalpolynoms p_n ist.

Beweis. Wir betrachten das Restglied für die *Hermite*-Interpolierende $h(x)$, welche die Werte und Ableitungen $f(x_1), f'(x_1), \ldots, f(x_n), f'(x_n)$ von f interpoliert (vgl. hierzu Satz 2.20 aus Kapitel 2). Der Interpolationsfehler ergibt sich nach Satz 2.16 aus selbigem Kapitel zu

$$f(x) - h(x) = \prod_{j=1}^n (x-x_j)^2 \cdot \frac{f^{(2n)}\big(\varrho(x)\big)}{(2n)!}, \quad x \in [a,b],\ \varrho(x) \in (a,b)\,.$$

Nun sind x_i die Nullstellen von $p_n(x)$ und $p_n \in \mathbb{P}_n$. Da nach Voraussetzung k_n der führende Koeffizient von p_n ist, folgt $(x-x_1)\cdots(x-x_n) = p_n(x)/k_n$ und somit auch

$$f(x) - h(x) = \frac{p_n^2(x)}{k_n^2} \frac{f^{(2n)}\big(\varrho(x)\big)}{(2n)!}\,.$$

Nach Definition von h hat $f-h$ an den Stellen x_i doppelte Nullstellen, so dass

$$(2n)!\,k_n^2 \frac{f(x)-h(x)}{p_n^2(x)} = f^{(2n)}\big(\varrho(x)\big)$$

auf ganz $[a,b]$ definiert und stetig ist. Da weiter $\omega(x)p_n^2(x) \geq 0$ für alle $x \in [a,b]$ gilt, lässt sich der Mittelwertsatz der Integralrechnung (Satz 5.28) anwenden, d.h.

$$\int_a^b \omega(x)(f(x)-h(x))\,dx = \int_a^b \omega(x)\frac{p_n^2(x)}{k_n^2}\frac{f^{(2n)}\big(\varrho(x)\big)}{(2n)!}\,dx = \frac{f^{(2n)}(\xi)}{(2n)!}\frac{(p_n,p_n)_\omega}{k_n^2} \quad \text{mit } \xi \in [a,b]\,.$$

Betrachtet man nun $h \in \mathbb{P}_{2n-1}$ mit $h(x_i) = f(x_i)$ und $h'(x_i) = f'(x_i)$ $(i = 1,\ldots,n)$, so folgt aus der Integrationseigenschaft der *Gauß*-Quadratur

$$\int_a^b \omega(x)f(x)\,dx - \sum_{i=1}^n w_i f(x_i) = \int_a^b \omega(x)f(x)\,dx - \sum_{i=1}^n w_i h(x_i)$$

$$= \int_a^b \omega(x)\big(f(x)-h(x)\big)\,dx = \frac{f^{(2n)}(\xi)}{(2n)!}\frac{(p_n,p_n)_\omega}{k_n^2},$$

womit die Behauptung bewiesen wäre. ■

Satz 5.30 Es sei $\omega \in C(a,b)$ eine positive Gewichtsfunktion auf (a,b) und $J_n = J_n(\omega) \in \mathbb{R}^{n\times n}$ wie in Definition 5.15 beschrieben. Des Weiteren sei $\mathcal{U}$ die Matrix, die spaltenweise die Eigenvektoren $u_k = (u_{k1},\dots,u_{kn})^T$ zu den Eigenwerten x_k $(k = 1,\dots,n)$ von J_n enthält, d.h.

$$J_n \mathcal{U} = \operatorname{diag}(x_1,\dots,x_n)\,\mathcal{U}\,.$$

Dann sind die Gewichte in Satz 5.25 gegeben durch

$$\omega_k = (1,1)_\omega \frac{u_{k1}^2}{u_k^T u_k} \quad (k = 1,\dots,n)\,. \tag{5.25}$$

Beweis. Nach Satz 5.15 sind die Eigenwerte $x_k^{(n)}$ von J_n die Nullstellen von $\tilde{p}_n(x)$ und $v_k := \tilde{\boldsymbol{p}}(x_k^{(n)}) = (\tilde{p}_0(x_k^{(n)}), \tilde{p}_1(x_k^{(n)}),\dots,\tilde{p}_{n-1}(x_k^{(n)}))^T$ zugehörige Eigenvektoren. Die Vektoren v_k sind Vielfache der Eigenvektoren u_k $(k = 1,\dots,n)$ mit der Eigenschaft, dass für den ersten Eintrag jeweils $v_{k1} = \tilde{p}_0$ gilt. Sei $V = (v_1|v_2|\cdots|v_n)$ die Matrix, die spaltenweise die Eigenvektoren v_k enthält und $\vec{\omega} = (\omega_1,\omega_2,\dots,\omega_n)^T$ der Vektor der gesuchten Gewichte. Aufgrund der verwendeten Normierung steht in der ersten Zeile jeweils $\tilde{p}_0$. Nach Satz 5.25 integriert die gesuchte Quadraturformel Polynome bis zum Grad $2n-1$ exakt, d.h.

$$\sum_{j=1}^n \omega_j q(x_j) = \int_a^b \omega(x) q(x) dx = \begin{cases} (1,\tilde{p}_0)_\omega & \text{für } q = \tilde{p}_0, \\ 0 & \text{für } q = \tilde{p}_k,\ 1 \le k \le n. \end{cases}$$

Damit folgt

$$V\,\vec{\omega} = \begin{pmatrix} \sum_k \omega_k \tilde{p}_0 \\ \sum_k \omega_k \tilde{p}_1(x_k^{(n)}) \\ \vdots \\ \sum_k \omega_k \tilde{p}_{n-1}(x_k^{(n)}) \end{pmatrix} = \tilde{p}_0 \begin{pmatrix} (1,1)_\omega \\ 0 \\ \vdots \\ 0 \end{pmatrix}. \tag{5.26}$$

Die Nullstellen von $\tilde{p}_n$ sind einfach nach Satz 5.23. Damit folgt aus Satz 5.15, dass auch die Eigenwerte von J_n einfach sind. Für Eigenvektoren v_k, v_m zu verschiedenen Eigenwerten $x_k^{(n)}$, $x_m^{(n)}$ $(k \neq m)$ gilt nun, da J_n symmetrisch ist

$$x_m^{(n)}\, v_m^T v_k = x_m^{(n)}\, v_k^T v_m = v_k^T J_n\, v_m = v_m^T J_n^T v_k = v_m^T J_n\, v_k = x_k^{(n)}\, v_m^T v_k\,.$$

Somit gilt $v_m^T v_k = 0$ für $k \neq m$, d.h. Eigenvektoren zu paarweise verschiedenen Eigenwerten stehen senkrecht aufeinander. Für einen beliebigen Eigenvektor v_k ergibt sich unter Ausnutzung von (5.26) und der eben gezeigten Orthogonalität

$$\tilde{p}_0^2(1,1)_\omega = \begin{pmatrix} \tilde{p}_0 \\ \tilde{p}_1(x_k^{(n)}) \\ \vdots \\ \tilde{p}_{n-1}(x_k^{(n)}) \end{pmatrix}^T \cdot \begin{pmatrix} \tilde{p}_0(1,1)_\omega \\ 0 \\ \vdots \\ 0 \end{pmatrix} = v_k^T \mathcal{V}\,\vec{\omega} = \widehat{v}_k^T \sum_{i=1}^n \omega_i v_i = \omega_k\, v_k^T\, v_k\,.$$

Berücksichtigung der vorangegangenen Normierung, d.h. $v_k = \tilde{p}_0 u_k / u_{k1}$, liefert

$$\omega_k = \frac{\tilde{p}_0^2(1,1)_\omega}{v_k^T\, v_k} = \frac{\tilde{p}_0^2(1,1)_\omega u_{k1}^2}{\tilde{p}_0^2 u_k^T u_k} = \frac{(1,1)_\omega u_{k1}^2}{u_k^T u_k}$$

und somit das Ergebnis. ∎

Tabelle 5.1.: Momente für einige Standardfälle von Orthogonalpolynomen

Name	a	b	$\omega(x)$	$m_k := \int_a^b \omega(x)x^k\,dx$
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 5.31 Sind die Momente $m_k := \int_a^b \omega(x)x^k\,dx$ $(k = 0, \ldots, 2n-1)$ bekannt, so lassen sich mit Satz 5.11 die Koeffizienten β_j $(j = 1, \ldots, n)$ und γ_j $(j = 2, \ldots, n)$ in der Drei-Term-Rekursion

$$p_n(x) = (x - \beta_n + x)p_{n-1}(x) - \gamma_n\, p_{n-2}(x)$$

mittels

$$\beta_n = \frac{(xp_{n-1}, p_{n-1})_\omega}{(p_{n-1}, p_{n-1})_\omega}, \qquad \gamma_n = \frac{(p_{n-1}, p_{n-1})_\omega}{(p_{n-2}, p_{n-2})_\omega}$$

bestimmen. Sei $p_k(x) = \sum_{j=0}^k \lambda_j x^j$ $(k \geq 0)$ gegeben, so erhält man

$$(p_k, p_k)_\omega = \sum_{i=0}^k \sum_{j=0}^k \lambda_i\lambda_j \int_a^b \omega(x)x^{i+j}\,dx = \sum_{i=0}^k \sum_{j=0}^k \lambda_i\lambda_j \cdot m_{i+j}$$

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 β_j, γ_j in die Tridiagonalmatrix $J_n(\omega)$ mit den Einträgen gegeben durch Definition 5.14 ein, so sind die Quadraturstellen zur Gewichtsfunktion ω gerade die Eigenwerte von $J_n(\omega)$ und die Gewichte lassen sich mittels (5.25) aus den zugehörigen Eigenvektoren bestimmen.

Die folgende Matlab-Routine setzt das in Bemerkung 5.31 beschriebene Vorgehen um.

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 - bb_(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 )
bb(1) = hx(1)/h(1);
p{1} = 1;
p{2} = [-bb(1);1];
for k = 3:n                         % best. p_k mit 3-Term-Rekur.
  q = conv(p{k-1},p{k-1});          % ber.  q = p_(k-1) * p_(k-1)
  h(k-1) = integ(q,momente);        % ber.  int_a^b omega*q dx
  hx(k-1) = integ([0;q],momente);% ber.  int_a^b omega*x*q dx
  bb(k-1,1) = hx(k-1)/h(k-1);       % best. Koeff. vor p_(k-1)
  cc(k-1,1) = h(k-1)/h(k-2);        % best. Koeff. vor p_(k-2)
  p{k} = [0;p{k-1}]-bb(k-1)*[p{k-1};0]-cc(k-1) * [p{k-2};0;0];
end
if length(bb)==1                    % sym. Matrix A aufstellen
  A = bb;
else
  A = full(spdiags([[sqrt(cc(2:end));0],bb,sqrt(cc)],...
      [-1 0 1],n-1,n-1));
end
[V,D] = eig(A);                     % Eigenwerte bestimmen
nodes = diag(D);                    % Quad.pkte EW von A
weights = zeros(n-1,1);             % Quad.qewichte skalierte
for k = 1:n-1                       % Skalarprodukte der EV
  weights(k) = momente(1) * V(1,k)^2 / (V(:,k)' * V(:,k));
end
kn = integ(conv(p{n},p{n}),momente); % Koeff. im Fehlerterm
% Integral auswerten
  function r = integ(p,momente)
    r = (p(:))'*momente(1:length(p))';
  end
end
```

Die nächste Routine verwendet die Momente aus Tabelle 5.1 und liefert Quadraturstellen, Quadraturgewichte und Fehlerkoeffizient $K_n = (p_n, p_n)_\omega / k_n^2$ in Satz 5.29 zu beliebigem $n \in \mathbb{N}$.

MATLAB-Funktion: GaussQuadratur.m

```
function [weights,nodes,kn] = GaussQuadratur(anz_Gewichte,typ)
  N = 2 * anz_Gewichte+1;          % Anzahl benoetiger Momente
  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" moeglich!')
      return
  end
  [weights,nodes,kn] = GaussWeightsNodes(momente);
end
```

Bemerkung 5.32 Umfangreiche Tabellen von Stützstellen und Gewichten zu verschiedenen Gewichtsfunktionen findet man z.B. in [**Abramowitz**] und [**Stroud**].

Bemerkung 5.33 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.

Das folgende Beispiel zeigt die Anwendung der Routine `GaussQuadratur.m`.

MATLAB-Beispiel:

Man kann z.B. die Quadraturstellen und Gewichte zur Gewichtsfunktion

$$\omega(x) = -\log(x)$$

und $n = 2$ berechnen lassen, sie erfüllen dann

```
>> [w,x,Kn] = GaussQuadratur(2,'log')
w =
    0.7185
    0.2815
x =
    0.1120
    0.6023
Kn =
  0.0029
```

$$-\int_0^1 \log(x) f(x)\, dx = \sum_{j=1}^{n} \omega_j f(x_j) + \frac{f^{(2n)}(\xi)}{(2n)!} K_n\,,$$

mit $K_n = (p_n, p_n)_\omega / k_n^2$.

Es folgen nun für spezielle Orthogonalpolynome, nämlich Tschebyscheff-, Legendre- und Jacobi-Polynome, Zusammenfassungen einiger ihrer Eigenschaften.

5.2. *Tschebyscheff*-Polynome

Die **Tschebyscheff**-Polynome T_n sind orthogonal bezüglich des Skalarprodukts

$$(f, g)_\omega := \int_{-1}^{1} \frac{f(x)g(x)}{\sqrt{1-x^2}} dx$$

und werden standardisiert durch $T_n(1) = 1$.

Die **Tschebyscheff**-Polynome besitzen die folgenden Eigenschaften:

(*i*) Sie erfüllen die Drei-Term-Rekursion

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

(Man gewinnt die Formeln mit Hilfe von Satz 5.11 für $x \in \mathbb{R}$.)

(*ii*) Sie haben stets ganzzahlige Koeffizienten.

(*iii*) Der höchste Koeffizient von T_n ist $a_n = 2^{n-1}$.

(*iv*) T_n ist stets eine gerade Funktion, falls n gerade, und eine ungerade, falls n ungerade ist.

(*v*) $T_n(1) = 1$, $T_n(-1) = (-1)^n$.
(*vi*) $|T_n(x)| \leq 1$ für $x \in [-1, 1]$.
(*vii*) Die Nullstellen von $T_n(x)$ sind

$$x_k := \cos\left(\frac{2k-1}{2n}\pi\right), \quad (k = 1, \ldots, n)\,.$$

(*viii*)

$$T_k(x) = \begin{cases} \cos(k \cdot \arccos(x)) & \text{, falls } |x| \leq 1\ , \\ \cosh(k \cdot \operatorname{Arccosh}(x)) & \text{, falls } x > 1\ , \\ (-1)^k \cosh(k \cdot \operatorname{Arccosh}(-x)) & \text{, falls } x < -1\ . \end{cases}$$

(*ix*) Die *Tschebyscheff*-Polynome besitzen die globale Darstellung

$$T_k(x) = \frac{1}{2}((x + \sqrt{x^2-1})^k + (x - \sqrt{x^2-1})^k), \qquad \text{wobei} \quad x \in \mathbb{R}.$$

(*x*) $|T_n(x)|$ nimmt seinen maximalen Wert im Intervall $[-1, 1]$ an den sogenannten *Tschebyscheff*-Abszissen $\overline{x}_k = \cos(\frac{k\pi}{n})$ für $k = 0, \ldots, n$ an, d.h.

$$|T_n(x)| = 1 \Leftrightarrow x = \overline{x}_k = \cos(\frac{k\pi}{n}) \quad \text{mit } k = 0, \ldots, n.$$

Abschließend möchten wir für $n = 0, \ldots, 5$ die T_n explizit angeben und diese auch anhand der nachfolgenden Grafik veranschaulichen:

$$\begin{aligned} T_0 &= 1\,, & T_3 &= 4x^3 - 3x\,, \\ T_1 &= x\,, & T_4 &= 8x^4 - 8x^2 + 1\,, \\ T_2 &= 2x^2 - 1\,, & T_5 &= 16x^5 - 20x^3 + 5x\,. \end{aligned}$$

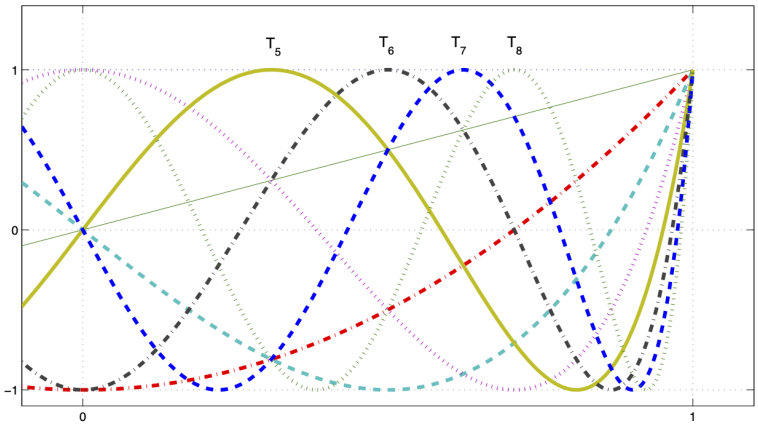


Abbildung 5.1.: Tschebyscheff-Polynome T_n $(n = 0, \ldots, 8)$.

5.3. *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.

$$(f, g)_\omega := \int_{-1}^{1} f(x)g(x)dx,$$

und sind durch $P_n(1) = 1$ standardisiert.

Die **Legendre**-Polynome besitzen folgende Eigenschaften:

(i) Sie erfüllen die Drei-Term-Rekursion

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

(ii) $P_n(1) = 1, P_n(-1) = (-1)^n \quad (n = 0, 1, 2, \ldots)$

(iii)

$$\int_{-1}^{1} P_n^2(x)\,dx = \frac{2}{2n+1} \quad (n = 0, 1, 2, \ldots) \tag{5.27}$$

(iv) Für Ableitung und Stammfunktion gilt

$$P_n'(x) = \frac{n(n+1)}{2n+1}\frac{P_{n+1}(x) - P_{n-1}(x)}{x^2 - 1} \quad (n \geq 1) \tag{5.28}$$

bzw.

$$\int_{-1}^{x} P_n(\xi)\,d\xi = \frac{1}{2n+1}\Big(P_{n+1}(x) - P_{n-1}(x)\Big) \quad (n \geq 1)\,. \tag{5.29}$$

Im Folgenden sind $P_0, \ldots, P_5$ explizit angegeben und in der nachfolgenden Grafik aufgezeichnet:

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

Aufgabe 5.34 (i) Man zeige für $m \geq 1$

$$\int_{-1}^{1}(1-x^2)P_m'(x)P_j(x)\,dx = \begin{cases} \frac{2m(m+1)}{(2m-1)(2m+1)} & \text{falls } j = m-1,\\ -\frac{2m(m+1)}{(2m+1)(2m+3)} & \text{falls } j = m+1,\\ 0 & \text{sonst.} \end{cases}$$

(ii) Man beweise (5.28) mit Hilfe von i).

(iii) Man zeige für $i, j \in \mathbb{N}_0$

$$\int_{-1}^{1} P_i(x)P_j'(x)\,dx = \begin{cases} 2 & \text{falls } i < j \text{ und } j+i \text{ ungerade,}\\ 0 & \text{sonst.} \end{cases}$$

(iv) Man beweise (5.29) mit Hilfe von iii).

(v) Man zeige für $n \in \mathbb{N}$

$$\int_{-1}^{x} P_n(\xi)\, d\xi = \frac{1}{n(n+1)} \left(x^2 - 1\right) P_n'(x)\,.$$

■

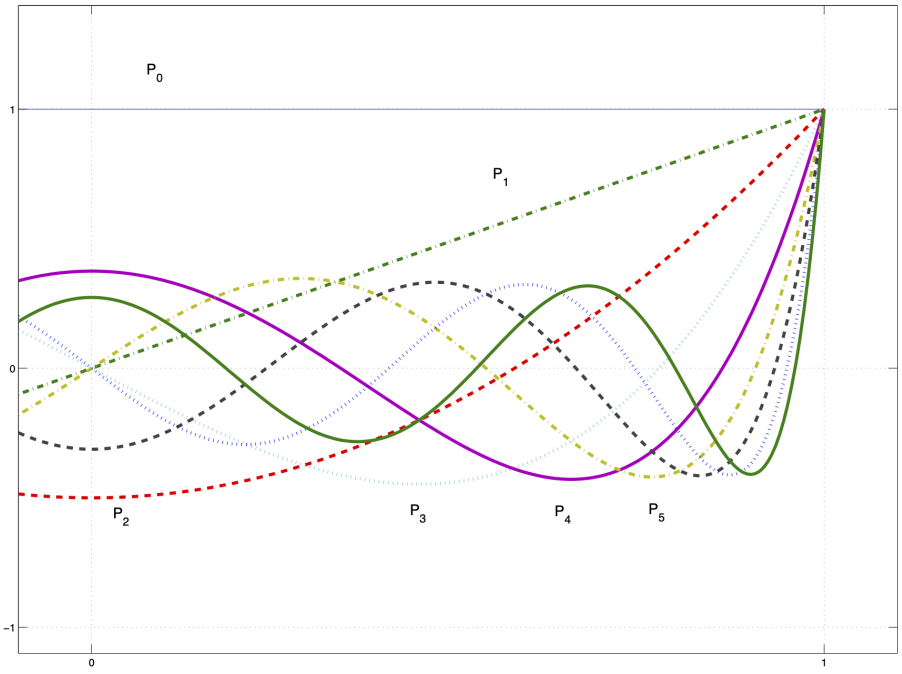


Abbildung 5.2.: Legendre-Polynome P_n $(n = 0, \ldots, 8)$.

5.4. *Jacobi*-Polynome

Die *Jacobi*-Polynome $P_n^{(\alpha,\beta)}$ sind orthogonal bezüglich des durch die positive Gewichtsfunktion
$(1-x)^\alpha(1+x)^\beta (\alpha > -1, \beta > -1)$ auf dem Intervall $(-1, 1)$ induzierten Skalarprodukts

$$(f, g)_\omega := \int_{-1}^{1} (1-x)^\alpha (1+x)^\beta f(x) g(x)\, dx.$$

Somit sind *Legendre*- und *Tschebyscheff*-Polynome spezielle *Jacobi*-Polynome (vgl. hierzu auch Bemerkung **??**), für die wir im Folgenden eine Aussage über die Verteilung der Nullstellen wiedergeben:

Satz 5.35 — (Szegoe). 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.

$$1 > x_1 > x_2 > \ldots > x_n > -1;\ 0 < \theta_1 < \theta_2 < \ldots < \theta_n < \pi\,.$$

Dann gilt

$$\frac{k + (\alpha+\beta-1)/2}{n + (\alpha+\beta+1)/2}\pi < \theta_k < \frac{k}{n + (\alpha+\beta+1)/2}\pi, \quad k = 1, 2, \ldots$$

und im Falle $\alpha = \beta$ gilt

$$\theta_k \geq \frac{k + \alpha/2 - 1/4}{n + \alpha + 1/2}\pi, \quad k = 1, 2, \ldots, [n/2],$$

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 **[Szegoe]**. ■

5.5. Linearkombination von Orthogonalpolynomen

Gegeben sei eine Linearkombination von Orthogonalpolynomen der Form

$$S_N(x) = \sum_{k=0}^{N} \alpha_k p_k(x),$$

wobei alle $p_k(x)$ einer Drei-Term-Rekursion der Form (5.12) mit den Startwerten p_0, p_1 genügen. Hierfür ergeben sich folgende Möglichkeiten zur Auswertung einer Linearkombination von Orthogonalpolynomen, nämlich Vorwärts- und Rückwärtsrekursion.

5.5.1. Vorwärtsrekursion

Eine naheliegende Berechnung der obigen Summe besteht darin, in jedem Schritt mit Hilfe von (5.12) die Werte p_k zu berechnen, diese mit den α_k zu multiplizieren und schließlich aufzuaddieren. Wir fassen im Folgenden die Summe $a_k + b_k x$ zum Faktor $\underline{a}_k$ zusammen, d.h.

$$p_k = \underline{a}_k p_{k-1} + \underline{b}_k p_{k-2}$$

Aus obigem Schema gewinnen wir folgenden Algorithmus für die Vorwärtsrekursion:

Seien p_0, p_1 und $s_1 := \alpha_0 p_0 + \alpha_1 p_1$ gegeben, dann gilt für $k = 2, \ldots, N$:

$$\begin{aligned} p_k &= \underline{a}_k p_{k-1} + \underline{b}_k p_{k-2} \\ s_k &= s_{k-1} + \alpha_k p_k \end{aligned}$$

Die Auswertung der $p = (p_0, \dots, p_N)$ ist nun äquivalent der Auswertung des gestaffelten Gleichungssystems

$$\underbrace{\begin{pmatrix} 1 & & & & \cdots & 0 \\ 0 & 1 & & & & \vdots \\ -\underline{b}_2 & -\underline{a}_2 & 1 & & & \\ & -\underline{b}_3 & -\underline{a}_3 & 1 & & \\ \vdots & & & \ddots & \ddots & \\ 0 & \cdots & & -\underline{b}_N & -\underline{a}_N & 1 \end{pmatrix}}_{=:\, L} \cdot \underbrace{\begin{pmatrix} p_0 \\ \vdots \\ \\ \\ \vdots \\ p_N \end{pmatrix}}_{=p} = \underbrace{\begin{pmatrix} p_0 \\ p_1 \\ 0 \\ \vdots \\ \\ 0 \end{pmatrix}}_{=:\, r} .$$

Die gegebene Linearkombination S_N ist also gerade das Skalarprodukt

$$S_N = \sum_{k=0}^{N} \alpha_k p_k = \langle \alpha, p \rangle \big(:= \alpha^T p \big), \quad \text{mit } Lp = r\,.$$

5.5.2. Rückwärtsrekursion Sei u Lösung von $L^T u = \alpha$, d.h.

$$S_N = \langle \alpha, L^{-1} r \rangle = \langle L^{-T} \alpha, r \rangle = \langle u, r \rangle,$$

bzw. in der Matrixschreibweise

$$\begin{pmatrix} 1 & 0 & -\underline{b}_2 & & \cdots & 0 \\ & 1 & -\underline{a}_2 & -\underline{b}_3 & & \vdots \\ & & 1 & -\underline{a}_3 & & \\ & & & 1 & \ddots & -\underline{b}_N \\ \vdots & & & & \ddots & -\underline{a}_N \\ 0 & \cdots & & & & 1 \end{pmatrix} \cdot \begin{pmatrix} u_0 \\ \vdots \\ \\ \\ \vdots \\ u_N \end{pmatrix} = \begin{pmatrix} \alpha_0 \\ \vdots \\ \\ \\ \vdots \\ \alpha_N \end{pmatrix}$$

Als resultierenden Algorithmus erhalten wir:

Es gelte $u_{N+1} = u_{N+2} = 0$ sowie $u_0 := b_2 u_2 + \alpha_0$, dann erhalten wir für $k = N, N-1, \dots, 1$:

$$u_k = \underline{a}_{k+1} u_{k+1} + \underline{b}_{k+1} u_{k+2} + \alpha_k$$

und $S_N := u_0 p_0 + u_1 p_1$.

Bemerkung 5.36 Wir wollen nun die beiden vorgestellten Algorithmen hinsichtlich ihrer Operationen betrachten:

- Vorwärtsrekursion: $k = 2, \dots, N$ jeweils 5 Operationen und 3 zu Beginn ergibt $5(N-1) + 3 = 5N - 2$ Operationen
- Rückwärtsrekursion: $k = N, \dots, 1$ jeweils 4 Operationen und 2 Operationen für u_0 bzw. 3 Operationen zur Berechnung von S_N im letzten Schritt ergeben insgesamt $4N + 5$

> Demzufolge spart man bei Verwendung der Rückwärtsrekursion $\approx N$ Operationen, d.h. man ist gegenüber der Vorwärtsrekursion um bis 20 Prozent schneller.

Anhang A.

Normen

Normen erfüllen den gleichen Zweck auf Vektorräumen, den Beträge auf der reellen Achse erfüllen. Genauer gesagt, $\mathbb{R}^n$ mit einer Norm auf $\mathbb{R}^n$ liefert einen metrischen Raum. Daraus ergeben sich bekannte Begriffe wie Umgebung, offene Menge, Konvergenz und Stetigkeit für Vektoren und vektorwertige Funktionen.

Definition A.1 — Vektornorm. Eine Vektornorm auf $\mathbb{R}^n$ ist eine Funktion $\|\cdot\| : \mathbb{R}^n \to \mathbb{R}$ mit den Eigenschaften

$$\begin{aligned} &\|x\| \geq 0 && x \in \mathbb{R}^n, \quad \|x\| = 0 \Leftrightarrow x = 0 \\ &\|x + y\| \leq \|x\| + \|y\| && x, y \in \mathbb{R}^n, \\ &\|\alpha x\| = |\alpha| \, \|x\| && \alpha \in \mathbb{R},\ x \in \mathbb{R}^n. \end{aligned} \tag{A.1}$$

Eine nützliche Klasse von Vektornormen sind die p-Normen, definiert durch

$$\|x\|_p = \left(\sum_{k=0}^{n} |x_k|^p \right)^{\frac{1}{p}} \tag{A.2}$$

Von diesen sind besonders die 1, 2 und ∞ interessant:

$$\begin{aligned} \|x\|_1 &= |x_1| + \ldots + |x_n| \\ \|x\|_2 &= (|x_1|^2 + \ldots + |x_n|^2)^{\frac{1}{2}} = (x^T x)^{\frac{1}{2}} \\ \|x\|_\infty &= \max_{1 \leq j \leq n} |x_j| \end{aligned} \tag{A.3}$$

Ein **Einheitsvektor** bzgl. der Norm $\|\cdot\|$ ist ein Vektor x mit $\|x\| = 1$.

A.1. Vektornorm-Eigenschaften

Ein klassisches Ergebnis bzgl. p-Normen ist die **Hölder-Ungleichung**

$$|x^T y| \leq \|x\|_p \, \|y\|_q \qquad \text{mit } \frac{1}{p} + \frac{1}{q} = 1. \tag{A.4}$$

Spezialfall der Hölder-Ungleichung ist die **Cauchy-Schwarz-Ungleichung**

$$|x^T y| \leq \|x\|_2 \, \|y\|_2. \tag{A.5}$$

Allen Normen auf $\mathbb{R}^n$ sind äquivalent, d.h. es seien $\|\cdot\|_\alpha$ und $\|\cdot\|_\beta$ Normen auf $\mathbb{R}^n$, dann existieren positive Konstanten c_1, c_2, so dass

$$c_1 \, \|x\|_\alpha \leq \|x\|_\beta \leq c_2 \, \|x\|_\alpha \qquad (x \in \mathbb{R}^n). \tag{A.6}$$

Zum Beispiel sei $x \in \mathbb{R}^n$, dann gilt

$$\begin{aligned} \|x\|_2 &\leq \|x\|_1 \leq \sqrt{n} \, \|x\|_2 \\ \|x\|_\infty &\leq \|x\|_2 \leq \sqrt{n} \, \|x\|_\infty \\ \|x\|_\infty &\leq \|x\|_1 \leq n \, \|x\|_\infty. \end{aligned} \tag{A.7}$$

Aufgabe A.2 Man beweise die Aussagen (A.4)–(A.7). ■

A.2. Matrixnormen

Da $\mathbb{R}^{m\times n}$ isomorph (d.h. es existiert eine bijektive Abb. mit $\varphi(\lambda A+\mu B) = \lambda\varphi(A)+\mu\varphi(B)$) zu $\mathbb{R}^{m\cdot n}$ ist, sollte die Definition einer Matrixnorm äquivalent sein zur Definition einer Vektornorm.

Definition A.3 — Matrixnorm. Es sei $\|\cdot\| : \mathbb{R}^{m\times n} \to \mathbb{R}$. Dann ist $\|\cdot\|$ eine Matrixnorm, wenn folgende Eigenschaften gelten:

$$\begin{aligned} &\|A\| \geq 0 && A \in \mathbb{R}^{m\times n}, \qquad \|A\| = 0 \Leftrightarrow A = 0) \\ &\|A + B\| \leq \|A\| + \|B\| && A, B \in \mathbb{R}^{m\times n} \\ &\|\alpha A\| = |\alpha| \, \|A\| && \alpha \in \mathbb{R}, A \in \mathbb{R}^{m\times n}. \end{aligned} \tag{A.8}$$

Bemerkung A.4 Die mit am häufigsten verwendeten Normen in der Numerischen Linearen Algebra sind die Frobenius-Norm

$$\|A\|_F := \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2} \tag{A.9}$$

und die p-Normen

$$\|A\|_p := \sup_{x\neq 0} \frac{\|Ax\|_p}{\|x\|_p} \, . \tag{A.10}$$

Man beachte folgende äquivalente Definition

$$\|A\|_p = \sup_{x\neq 0} \left\| A \frac{x}{\|x\|} \right\|_p = \sup_{\|x\|_p=1} \|Ax\|_p \,. \tag{A.11}$$

Bemerkung A.5 — Submultiplikativität. Die Frobenius-Norm und die p-Normen erfüllen zusätzlich auch noch die Eigenschaft der **Submultiplikativität**, d.h.

$$\|A \cdot B\| \leq \|A\| \, \|B\|. \tag{A.12}$$

Bemerkung A.6 Es sei y ein Vektor mit $\|y\|_p = 1$, dann gilt

$$\begin{aligned} \|A \cdot B\|_p &= \max_{x\neq 0} \frac{\|ABx\|_p}{\|x\|_p} = \max_{x\neq 0} \frac{\|ABx\|_p \, \|Bx\|_p}{\|Bx\|_p \, \|x\|_p} \\ &\leq \max_{x\neq 0} \frac{\|ABx\|_p}{\|Bx\|_p} \cdot \max_{x\neq 0} \frac{\|Bx\|_p}{\|x\|_p} \\ &= \max_{y\neq 0} \frac{\|Ay\|_p}{\|y\|_p} \cdot \max_{x\neq 0} \frac{\|Bx\|_p}{\|x\|_p} = \|A\|_p \, \|B\|_p \,. \end{aligned}$$

Bemerkung A.7 i) Nicht alle Matrix-Normen erfüllen diese Eigenschaft, z.B. gilt für $\|A\|_A := \max |a_{ij}|$ und $A = B = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$ die Ungleichung

$$\|A \cdot B\|_A = 2 > \|A\|_A \, \|B\|_A = 1 \,.$$

ii) Für die p-Normen haben wir die wichtige Eigenschaft, dass für alle $A \in \mathbb{R}^{m\times n}$ und $x \in \mathbb{R}^n$

$$\|Ax\|_p \leq \|A\|_p \, \|x\|_p. \tag{A.13}$$

Die Frobenius und p-Normen (speziell $p = 1, 2, \infty$) erfüllen gewisse Ungleichungen, welche in der Analysis von Matrixberechungen verwendet werden. Es sei $A \in \mathbb{R}^{m\times n}$, dann gilt

$$\|A\|_2 \leq \|A\|_F \leq \sqrt{n} \, \|A\|_2 \tag{A.14}$$

$$\max_{i,j} |a_{ij}| \leq \|A\|_2 \leq \sqrt{m\,n} \, \max_{i,j} |a_{ij}| \tag{A.15}$$

$$\frac{1}{\sqrt{n}} \, \|A\|_\infty \leq \|A\|_2 \leq \sqrt{m} \, \|A\|_\infty \tag{A.16}$$

$$\frac{1}{\sqrt{m}} \, \|A\|_1 \leq \|A\|_2 \leq \sqrt{n} \, \|A\|_1 \tag{A.17}$$

Des Weiteren gilt für $A \in \mathbb{R}^{m\times n}$

$$\begin{aligned}\|A\|_\infty &= \max_{\|x\|_\infty=1} \|Ax\|_\infty = \max_{\|x\|_\infty=1} \left\{ \max_{1\le j\le m} \left| \sum_{k=1}^{n} a_{jk}x_k \right| \right\} \\ &= \max_{1\le j\le m} \left\{ \max_{\|x\|_\infty=1} \left| \sum_{k=1}^{n} a_{jk}x_k \right| \right\} = \max_{1\le j\le m} \sum_{k=1}^{n} |a_{jk}|.\end{aligned}$$

Aufgrund dieser Gleichung bezeichnet man $\|\cdot\|_\infty$ auch als **Zeilensummennorm**. Diese ist mittels der letzten Gleichung auch leicht zu bestimmen, d.h.

$$\|A\|_\infty = \max_{1\le j\le m} \sum_{k=1}^{n} |a_{jk}| \,. \tag{A.18}$$

Analog gilt für die 1-Norm:

$$\begin{aligned}\|A\|_1 &= \max_{\|x\|_1=1} \|Ax\|_1 = \max_{\|x\|_1=1} \sum_{j=1}^{m} \left| \sum_{k=1}^{n} a_{jk}x_k \right| = \max_{\|x\|_1=1} \sum_{j=1}^{m}\sum_{k=1}^{n} |a_{jk}|\,|x_k|\,\operatorname{sign}(a_{jk}x_k) \\ &= \max_{\|x\|_1=1} \sum_{k=1}^{n} |x_k| \sum_{j=1}^{m} |a_{jk}|\,\operatorname{sign}(a_{jk}x_k) = \max_{\|x\|_1=1} \sum_{k=1}^{n} |x_k| \sum_{j=1}^{m} |a_{jk}| = \max_{1\le k\le m} \sum_{j=1}^{n} |a_{jk}| \,.\end{aligned}$$

Also gilt:

$$\|A\|_1 = \max_{1\le k\le n} \sum_{j=1}^{m} |a_{jk}| \,. \tag{A.19}$$

Diese Norm bezeichnet man auch als **Spaltensummennorm**.

Bemerkung A.8 Einfache Eselbrücke: $\boxed{1}$-Spaltensummen, $\boxed{\infty}$-Zeilensummen.

Auch für die 2-Norm lässt sich eine äquivalente Formulierung finden.

Satz A.9 Es sei $A \in \mathbb{R}^{m\times n}$. Dann existiert ein Vektor $z \in \mathbb{R}^n$ mit $\|z\|_2 = 1$, so dass

$$A^TAz = \mu^2 z \,, \quad \text{wobei } \mu = \|A\|_2 \,.$$

Bemerkung A.10 Der Satz impliziert, dass $\|A\|_2^2$ eine Nullstelle des Polynoms $p(z) = \det(A^TA - \lambda I)$ ist. Genauer betrachtet, ist die 2-Norm von A die Wurzel des größten Eigenwerts von A^TA.

Beweis. Es sei $z \in \mathbb{R}^n$ mit $\|z\|_2 = 1$ und $\|Az\|_2 = \|A\|_2$. Da z die Funktion

$$g(x) = \frac{1}{2}\frac{\|Ax\|_2^2}{\|x\|_2^2} = \frac{1}{2}\frac{x^TA^TAx}{x^Tx} \tag{A.20}$$

maximiert, folgt daraus, dass er $\nabla g(z) = 0$ erfüllt, wobei ∇g der Gradient von g ist $\left(\nabla := \left(\frac{\partial}{\partial x_1}, \frac{\partial}{\partial x_2}, \ldots, \frac{\partial}{\partial x_n}\right)\right)^T$. Die partiellen Ableitungen von g lauten für $i = 1, \ldots, n$

$$\frac{\partial g(z)}{\partial x_i} = \left[x^T x \sum_{j=1}^{n} (A^T A)_{ij} x_j - (x^T A^T A x) x_i \right] / (x^T x)^2. \tag{A.21}$$

In Vektornotation bedeutet dies für z, da $\nabla g(z) = 0$ und $\|z\|_2 = 1$, dass $A^T A z = (z^T A^T A z) z$. Setzt man nun $\mu = \|A\|_2$ so ergibt sich daraus die Behauptung. ■

Lemma A.11 Es sei $A \in \mathbb{R}^{m \times n}$. Dann gilt

$$\|A\|_2 \leq \sqrt{\|A\|_1 \, \|A\|_\infty}\,. \tag{A.22}$$

Beweis. Es sei $z \in \mathbb{R}^n$ mit $A^T A z = \mu^2 z$ und $\mu = \|A\|_2$, d.h. es sei $z \neq 0$ ein Vektor, für den das Maximum von $\dfrac{\|Ax\|_2}{\|x\|_2}$ angenommen wird. Dann gilt

$$\mu^2 \|z\|_1 = \|A^T A z\|_1 \leq \|A^T\|_1 \, \|Az\|_1 \leq \|A\|_\infty \, \|A\|_1 \, \|z\|_1\,. \tag{A.23}$$

Kondensation von $\|z\|_1$ und Wurzelziehen liefert das gewünschte Ergebnis. ■

Definition A.12 — verträgliche Vektornorm. Eine Matrixnorm $\|A\|$ heißt kompatibel oder verträglich mit der Vektornorm $\|x\|$, falls folgende Ungleichung erfüllt ist

$$\|Ax\| \leq \|A\| \, \|x\|\,, \qquad x \in \mathbb{R}^n, A \in \mathbb{R}^{m \times n}\,. \tag{A.24}$$

Kombinationen von verträglichen Normen sind etwa

$\|A\|_G, \|A\|_\infty$ sind verträglich mit $\|x\|_\infty$;
$\|A\|_G, \|A\|_1$ sind verträglich mit $\|x\|_1$;
$\|A\|_G, \|A\|_F, \|A\|_2$ sind verträglich mit $\|x\|_2$, wobei

$$\|A\|_G := n \max_{i,j} |a_{ij}| \qquad (A \in \mathbb{R}^{m \times n})\,.$$

Bemerkung A.13 Man verifiziere selbständig an einigen Beispielen die Verträglichkeit von o.g. Normenpaaren.

Abschließend erhalten wir am Ende des Kapitels.

Satz A.14 Die Matrix-p-Normen sind unter allen mit der Vektornorm $\|x\|_p$ verträglichen Matrixnormen die kleinsten.

Anhang B.

Einführung in MATLAB

B.1. Grundlegende MATLAB-Befehle

Ruft man das Programm MATLAB mit dem Befehl `matlab` auf, so erscheinen auf dem Monitor einige Fenster. Auf den Linux-Rechnern des KIZ müssen Sie vorher die notwendigen Pfade ergänzen. Geben Sie dazu in einem Konsole-Fenster den Befehl option matlab ein. Von diesen ist das Befehl-Fenster der primäre Ort um Befehle einzugeben und Fehler oder Ergebnisse abzulesen! Das Prompt-Zeichen ≫ ist im Befehl-Fenster dargestellt und dort findet man üblicherweise einen blinkenden Cursor. Der blinkende Cursor und der MATLAB-Prompt zeigen einem, dass MATLAB eine Eingabe erwartet.

B.1.1. Einfache mathematische Operationen

Genauso wie mit einem simplen Taschenrechner kann man auch mit MATLAB einfache mathematische Operationen ausführen, z.B. ergibt die Eingabe

```
≫ 3 + 4
```

die Ausgabe

```
ans =
     7
```

Man beachte, dass MATLAB im Allgemeinen keine Zwischenräume benötigt, um Befehle eindeutig zu verstehen. Alternativ zu dem obigen Beispiel können in MATLAB auch Variablen verwendet werden.

```
≫ a = 3
a =
      3
≫ b = 4
b =
      4
≫ c = a + b
c =
      7
```

MATLAB besitzt folgende einfache arithmetische Operationen

Operation	Symbol	Beispiel
Addition, $a + b$	+	5 + 3
Subtraktion, $a - b$	−	23−12
Multiplikation, $a \cdot b$	*	13.3 * 63.13
Division, $a$div b	/ or \	$17/4 = 4\backslash 17$
Potenz, a^b	^	3^4

Die Reihenfolge, in der eine Folge von Operationen abgearbeitet wird, lässt sich wie folgt beschreiben. Ausdrücke werden von links nach rechts ausgeführt, wobei die Potenzierung die höchste Priorität besitzt gefolgt von Punktoperation, sprich Multiplikation und Division. Die geringste Priorität haben Addition und Subtraktion. Mit Hilfe von Klammern kann diese Vorgehensweise geändert werden, wobei innere Klammern vor äußeren Klammern berechnet werden.

B.1.2. Variablen Wie in anderen Programmiersprachen hat auch MATLAB Regeln für Variablennamen. Eine Variable repräsentiert ein Datenelement, dessen Wert während der Programmausführung – gegebenenfalls mehrfach – geändert werden kann. Variablen werden anhand ihrer „Namen“ identifiziert. Namen bestehen aus ein bis neunzehn Buchstaben, Ziffern oder Unterstrichen, wobei das erste Zeichen ein Buchstabe sein muss. M an beachte, dass MATLAB Groß- und Kleinschreibung unterscheidet. (Windows ist im Gegensatz zu Linux nicht so restriktiv, da Sie jedoch Programme austauschen wollen, sollten Windows-Benutzer besondere Aufmerksamkeit walten lassen.)

Einer Variablen ist Speicherplatz zugeordnet. Wenn man eine Variable verwendet, dann meint man damit entweder den zugeordneten Speicherplatz oder den Wert, der dort augenblicklich abgespeichert ist. Einen Überblick über alle Variablen erhält man mit dem Befehl `who` oder `whos`, wobei letzterer die Angabe des benutzten Speicherplatzes beinhaltet.

Zusätzlich zu selbstdefinierten Variablen gibt es in MATLAB verschiedene spezielle Variablen. Diese lauten

spezielle Variablen	Wert
ans	standard Variablenname benutzt für Ergebnisse
pi	3.1415 . . .
eps	Maschinengenauigkeit
flops	Zähler für die Anzahl der Fließkommaoperationen
inf	steht für Unendlich (eng. infinity). z.B. 1/0
NaN	eng. Not a Number, z.B. 0/0
i (und) j	$i = j = \sqrt{-1}$

In MATLAB kann der Speicherplatz, der durch Variablen belegt ist, durch den Befehl `clear` wieder freigegeben werden, z.B.

```
≫ clear a b c
```

B.1.3. Kommentare und Punktion Der Text, der nach einem Prozentzeichen % folgt, wird in MATLAB als Kommentar verstanden

```
≫ dummy = 4 % Wert von dummy
dummy =
      4
```

Mehrere Befehle können in eine Zeile geschrieben werden, wenn sie durch Kommata oder Semikola getrennt werden. Kommata veranlassen MATLAB, die Ergebnisse anzuzeigen. Bei einem Semikolon wird die Ausgabe unterdrückt. Durch eine Sequenz von drei Punkten kann man einen Befehl in der folgenden Zeile fortsetzen.

B.1.4. Spezielle Funktionen Eine unvollständige Liste von Funktionen, die MATLAB bereitstellt, ist im folgenden dargestellt. Die meisten Funktionen sind so definiert, wie man sie üblicherweise benutzt.

```
≫ y = cos(pi)
y =
     -1
```

MATLAB bezieht sich im Zusammenhang mit Winkelfunktionen auf das Bogenmaß.

Funktion	Bedeutung
abs(x)	Absolutbetrag
cos(x)	Kosinus
exp(x)	Exponentialfunktion: e^x
fix(x)	rundet auf die nächste, vom Betrag her kleinere ganze Zahl
floor(x)	rundet auf die nächste, kleinere ganze Zahl
gcd(x,y)	größter gemeinsamer Teiler von x und y
lcm(x,y)	kleinstes gemeinsames Vielfaches von x und y
log(x)	natürlicher Logarithmus
rem(x,y)	Modulo (eng. remainder of division), z.B. rem(5,2)=1
sign(x)	Signum Funktion, z.B. sign(2.3) = 1, sign(0) = 0, sign(-.3) = -1
sin(x)	Sinus
sqrt(x)	Quadratwurzel
tan(x)	Tangens

B.1.5. Skript-Dateien Für einfache Probleme ist es schnell und effizient, die Befehle am MATLAB-Prompt einzugeben. Für größere und umfangreichere Aufgabenstellungen bietet MATLAB die Möglichkeit, sogenannte Skript-Dateien zu verwenden, in denen die Befehle in Form einer Textdatei aufgeschrieben sind und die man am Prompt übergibt. MATLAB öffnet dann diese Dateien und führt die Befehle so aus, als hätte man sie am Prompt eingegeben. Die Datei nennt man Skript-Datei oder M-Datei, wobei der Ausdruck M-Datei daher rührt, dass diese Dateien das Suffix `.m` haben, z.B. `newton.m`. Um eine M-Datei zu erstellen, ruft man einen Editor auf und speichert die Datei in dem Verzeichnis, von dem aus man MATLAB gestartet hat oder starten wird. Die Datei, z.B. `newton.m`, wird in MATLAB dann durch Eingabe von `newton` am Prompt aufgerufen.

Für die Benutzung von Skript-Dateien hat MATLAB unter anderem folgende hilfreichen Befehle

M-Datei-Funktionen	
disp(ans)	zeigt den Wert der Variablen `ans`, ohne ihren Namen auszugeben
input	erwartet vom Benutzer eine Eingabe
keyboard	übergibt zeitweise die Kontrolle an die Tastatur
pause	hält das Programm an, bis eine Taste betätigt wird

Die folgende Skript-Datei `beispiel1.m`

```
% beispiel1.m
% Beispiel fuer eine Skript-Datei
tmp = input(' Geben Sie bitte eine Zahl an >' );
3 * tmp;
```

führt zu der Ausgabe

```
≫ beispiel1
Geben Sie bitte eine Zahl an > 6
ans =
      18
```

B.1.6. Dateiverwaltung MATLAB unterstützt eine Vielzahl von Dateiverwaltungsbefehlen, welche es einem ermöglichen Dateien zu listen, Skript-Dateien anzusehen oder zu löschen und Verzeichnisse zu wechseln.

Datei-Managment-Funktionen	
cd path	wechselt in das Verzeichnis `path`
delete beispiel	löscht die Datei `beispiel.m`
ls	zeigt alle Dateien im aktuellen Verzeichnis an
pwd	zeigt den aktuellen Verzeichnispfad an
type beispiel	zeigt den Inhalt der Datei `beispiel.m` im Befehl-Fenster
what	zeigt alle M- und MAT-Dateien im aktuellen Verzeichnis an

B.1.7. Hilfe

Online-Hilfe: Da sich nicht jeder Benutzer alle MATLAB-Befehle merken kann oder auch von einigen auch nur die Syntax unklar ist, bietet MATLAB die Möglichkeit der Online-Hilfe. Dabei gibt es prinzipiell mehrere Möglichkeiten. Ist einem ein Befehl bekannt und man sucht Informationen über die Syntax, so gibt es den Befehl `help`.

```
≫ help sqrt
  SQRT   Square root.
         SQRT(X) is the square root of the elements of X. Complex
         results are produced if X is not positive.

         See also SQRTM.
```

Als Beispiel haben wir uns hier die Hilfe zu dem Befehl `sqrt` ausgeben lassen.

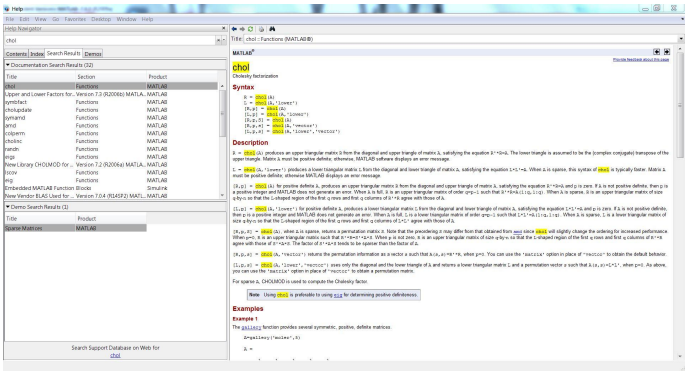
Die andere Möglichkeit der von MATLAB gelieferten Hilfe ist durch den Befehl `lookfor` gegeben. Hier durchsucht das Programm alle ersten Zeilen der MATLAB Hilfe-Kennwörter und Skript-Dateien die im MATLAB Suchpfad zu finden sind. Das Bemerkenswerte dabei ist, dass dieser Begriff kein Befehl zu sein braucht.

```
≫ lookfor cholesky
  CHOL    Cholesky factorization
≫  CHOL   Cholesky factorization.
  CHOL(X) uses only the diagonal and upper triangle of X.
  The lower triangular is assumed to be the (complex conjugate)
  transpose of the upper.  If X is positive definite, then
  R = CHOL(X) produces an upper triangular R so that R'*R =X.
  If X is not positive definite, an error message is printed.

  With two output arguments, [R,p] = CHOL(X) never produces an
  error message.  If X is positive definite, then p is 0 and R
  is the same as above.   But if X is not positive definite, then
  p is a positive integer and R is an upper triangular matrix of
  order q = p-1 so that R'*R = X(1:q,1:q).
```

```
≫ lookfor  factorization

CHOL    Cholesky factorization.
QRDELETE Delete a column from the QR factorization.
QRINSERT Insert a column in the QR factorization.
SYMBFACT Symbolic factorization analysis.
```

Eine weitere Möglichkeit, sich Hilfe zu verschaffen, besteht darin, das Helpdesk aufzurufen. Wenn Sie helpdesk am Prompt eingeben, öffnet sich die folgende Hilfsumgebung

Notiz



B.2. Mathematik mit Matrizen

B.2.1. Matrixkonstruktion und Adressierung Beschäftigen wir uns nun mit Möglichkeiten Matrizen in MATLAB zu definieren.

einfache Matrix Konstruktionen	
x=[1 4 2*pi 4]	erstelle einen Zeilenvektor x mit Einträgen
x=$anfang$:$ende$	erstelle einen Zeilenvektor x beginnend mit $anfang$, Inkrement 1 und endend mit $ende$
x=$anfang$:$inkrement$:$ende$	Ähnliches wie oben mit dem Inkrement $inkrement$
x=linspace($anfang$,$ende$,n)	erzeugt einen Zeilenvektor der Dimension n mit $x(i) = \frac{(n-i)\cdot anfang+(i-1)\cdot ende}{n-1}$

Im Folgenden sind einige charakteristische Beispiele aufgeführt.

```
≫ B = [1 2 3 4; 5 6 7 8]
B =
     1 2 3 4
     5 6 7 8
```

Der Operator ′ liefert für reelle Matrizen die Transponierte.

```
≫ C = B'
C =
     1 5
     2 6
     3 7
     4 8
```

Der Doppelpunkt : in der zweiten Komponente spricht alle vorhandenen Spalten an, d.h. er ist ein zu `1:4` äquivalenter Ausdruck.

```
≫ C = B(1,:)
C =
     1 2 3 4
≫C = B(:,3)'
C =
     3 7
```

Es lassen sich auch einzelne Komponenten neu definieren.

```
≫A = [1 2 3; 4 5 6; 7 8 9]
A =
     1 2 3
     4 5 6
     7 8 9
≫ A(1,3) = 9
A =
     1 2 9
     4 5 6
     7 8 9
```

Ist ein Eintrag noch nicht definiert, so verwendet MATLAB die minimale Erweiterung dieser Matrix und setzt undefinierte Einträge zu Null.

```
≫ A(2,5) = 4
A =
     1 2 9 0 0
     4 5 6 0 4
     7 8 9 0 0
```

Im Folgenden werden die Vektoren `(3,2,1)` und `(2,1,3,1,5,2,4)` dazu verwendet, die Matrix `C` zu indizieren, d.h. `C` hat die Struktur

```
A(3,2) A(3,1) A(3,3) A(3,1) A(3,5) A(3,2) A(3,4)
A(2,2) A(2,1) A(2,3) A(2,1) A(2,5) A(2,2) A(2,4)
A(1,2) A(1,1) A(1,3) A(1,1) A(1,5) A(1,2) A(1,4)
```

In MATLAB erhält man nun

```
≫ C=A(3:-1:1,[2 1 3 1 5 2 4])
C =
     8 7 9 7 0 8 0
     5 4 6 4 4 5 0
     2 1 9 1 0 2 0
```

Ein weiteres Beispiel für Indizierung ist

```
≫ C=C(1:2,2:3)
C =
     7 9
     4 6
```

Im nächsten Beispiel wird ein Spaltenvektor dadurch konstruiert, dass alle Elemente aus der Matrix `C` hintereinander gehängt werden. Dabei wird spaltenweise vorgegangen.

```
≫ b=C(:)'
b =
     7 4 9 6
```

Das Löschen einer ganzen Zeile oder Spalte kann durch das Umdefinieren in eine 0×0-Matrix geschehen, z.B.

```
≫ C(2,:)=[ ]
C =
     7 9
```

B.2.2. Skalar-Matrix-Operationen In MATLAB sind Skalar-Matrix-Operationen in dem Sinne definiert, dass Addition, Subtraktion, Division und Multiplikation mit einem Skalar elementweise durchgeführt werden. Es folgen zwei erklärende Beispiele.

```
≫ B - 1
ans =
     0 1 2 3
     4 5 6 7
≫ 9 + 3 * B
ans =
     12 15 18 21
     24 27 30 33
```

B.2.3. Matrix-Matrix-Operationen Die Operationen zwischen Matrizen sind nicht so kanonisch zu definieren wie die zwischen Skalar und Matrix, insbesondere sind Operationen zwischen Matrizen unterschiedlicher Dimension schwer zu definieren. Des Weiteren sind die Operationen $*$ und $.*$, bzw. / und ./ sowie \ und .\ zu unterscheiden. In nachfolgender Tabelle sind die Matrixoperationen beschrieben.

	komponentenweise Matrixoperationen
Beispieldaten	$a = [a_1, a_2, \ldots, a_n]$, $b = [b_1, b_2, \ldots, b_n]$, c ein Skalar
komp. Addition	$a + c = [a_1 + c\ a_2 + c \ldots a_n = c]$
komp. Multiplikation	$a * c = [a_1 \cdot c\ a_2 \cdot c\ \ldots\ a_n \cdot c]$
Matrix–Addition	$a + b = [a_1 + b_1\ a_2 + b_2 \ldots a_n + b_n]$
komp. Matrix-Multiplikationen	$a. * b = [a_1 \cdot b_1\ a_2 \cdot b_2 \ldots a_n \cdot b_n]$
komp. Matrix-Div. von rechts	$a./b = [a_1/b_1\ a_2/b_2 \ldots a_n/b_n]$
komp. Matrix-Div. von links	$a.\backslash b = [b_1/a_1\ b_2/a_2 \ldots b_n/a_n]$
komp. Matrix-Potenz	$a.\hat{}\,c = [a_1^c\ a_2^c \ldots a_n^c]$
	$c.\hat{}\,a = [c^{a_1}\ c^{a_2} \ldots c^{a_n}]$
	$a.\hat{}\,b = [a_1^{b_1}\ a_2^{b_2} \ldots a_n^{b_n}]$

Es folgen nun einige Beispiele zu Matrixoperationen

```
≫ g=[1 2 3; 4 5 6]; % zwei neue Matrizen
≫ h=[2 2 2; 3 3 3];
≫ g+h   % addiere g und h komponentenweise
ans =
     3  4  5
     7  8  9
≫ ans-g   % subtrahiere g von der vorherigen Antwort
ans =
     2  2  2
     3  3  3
≫ h.*g   % multipliziere g mit h komponentenweise
ans =
      2   4   6
     12  15  18
≫ g*h'   % multipliziere g mit h'
ans =
     12  18
     30  45
```

B.2.4. Matrix-Operationen und -Funktionen Nun einige Operationen die sich auf Matrizen anwenden lassen.

Matrixfunktionen	
reshape(A,m,n)	erzeugt aus den Eintägen der Matrix A eine $m \times n$-Matrix, wobei die Einträge spaltenweise aus A gelesen werden.
diag(A)	ergibt die Diagonale von A als Spaltenvektor
diag(v)	erzeugt Diagonalmatrix mit dem Vektor v in der Diagonalen
tril(A)	extrahiert den unteren Dreiecksanteil der Matrix A
triu(A)	extrahiert den oberen Dreiecksanteil der Matrix A

Es folgen einige Beispiele

```
≫ g = linspace(1,9,9)   % ein neuer Zeilenvektor
g =
     1    2    3    4    5    6    7    8    9
≫ B = reshape(g,3,3)   % macht aus g eine 3 x 3 Matrix
B =
     1    4    7
     2    5    8
     3    6    9
≫ tril(B)
ans =
     1    0    0
     2    5    0
     3    6    9
```

Funktion	Bedeutung
R=chol(A)	Choleskyzerlegung
cond(A)	Konditionszahl der Matrix A
d=eig(A)	Eigenwerte und -vektoren
[V,d]=eig(A)	
det(A)	Determinante
hess(A)	Hessenbergform
inv(A)	Inverse
[L,U]=lu(A)	Zerlegung gegeben durch Gauss-Algorithmus
norm(A)	euklidische-Norm
rank(A)	Rang der Matrix A

Bemerkung: Der \ Operator ist auch für Matrizen definiert und liefert in Kombination mit Vektoren für reguläre Matrizen ihre Inverse, d.h. `A^(-1)x=A\x` .

B.2.5. Spezielle Matrizen Einige häufig auftretende spezielle Matrizen sind im folgenden aufgelistet.

spezielle Matrizen	
eye(n)	erzeugt eine Einheitsmatrix der Dimension n
ones(m,n)	erzeugt eine $m \times n$-Matrix mit den Einträgen 1
zeros(m,n)	erzeugt eine $m \times n$-Matrix mit den Einträgen 0

B.2.6. Spezielle Funktionen für schwachbesetzte Matrizen Bei vielen numerischen Anwendungen treten schwachbesetzte Matrizen auf. MATLAB hat für solche Matrizen besondere Sparse-Funktionen, die dieser Eigenschaft Rechnung tragen.

Funktion	Bedeutung
find(A)	findet Indizes von Nichtnulleinträgen
nnz(A)	Anzahl an Nichtnulleinträgen
spdiags(v)	erzeugt eine Sparse-Diagonalmatrix mit Vektor v als Diagonale
speye(n)	erzeugt eine Sparse-Einheitsmatrix
spy(A)	visualisiert die Struktur der Matrix A

Kurze Illustration der Funktionsweise obiger Befehle anhand einiger Beispiele.

```
≫ E=eye(100);    % vollbesetzte 100 x 100 Einheitsmatrix
≫ Es=sparse(E);  % Sparse-Version von E
≫ whos
      Name      Size    Elements  Bytes  Density  Comple
         E  100 by 100    10000  80000     Full      No
        Es  100 by 100      100   1600   0.0100      No
 Grand total is 10100 elements using 81600 bytes
≫ A=spdiags([7*ones(4,1),ones(4,1),2*ones(4,1)],[-1,0,1],4,4);
≫ nnz(A)
ans =
     10
≫ full(A)
ans =
      1   2   0   0
      7   1   2   0
      0   7   1   2
      0   0   7   1
```

Als Beispiel für `spy` sei hier die Besetzungstruktur einer 3D-FEM Steifigskeitsmatrix in Abb. B.1 gezeigt.

B.3. Datenverwaltung

Für die meisten Anwendungen genügt es, Datenfelder in einem Format abzuspeichern und wieder laden zu können. Die Befehle `load` und `save` setzen voraus, dass die Daten in einem

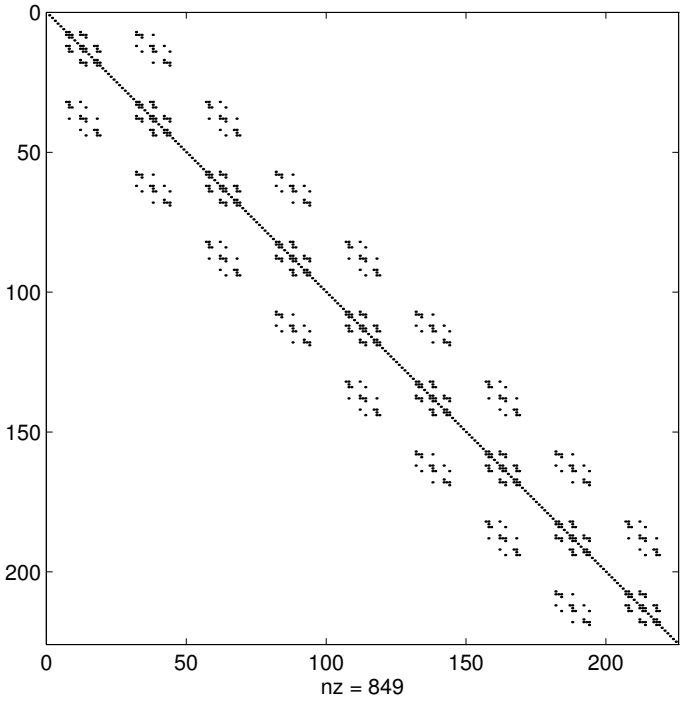


Abbildung B.1.: Besetzungstruktur einer 3D-FEM Steifigskeitsmatrix

System unabhängigen, binären Format in einer Datei mit dem Suffix `.mat` gespeichert sind oder in einem einfachen ASCII-Format vorliegen.

B.3.1. Daten speichern Im Folgenden wird eine 3×5 -Matrix im binär-Format in der Datei `A.mat` gespeichert. Diese Daten sind sehr kompakt gespeichert.

```
≫ A=zeros(3,5);
≫ save A
```

Gibt man sich aber den Inhalt dieser Datei auf dem Bilschirm aus, so gibt er wenig Sinn. Möchte man sich also z.B. einen Lösungsvektor sichern um ihn später „per Hand zu analysieren“ so speichere man die Daten als ASCII-Datei, dabei kann man wählen zwischen einer 8-stelligen oder 16-stelligen Abspeicherung.

```
≫ save mat1.dat A -ascii          %  8-stellige Speicherung
≫ save mat2.dat A -ascii -double  % 16-stellige Speicherung
```

Hier wurde die Matrix mit 8-stelligem Format in der Datei `mat1.dat` gespeichert, bzw. 16-stellig in der Datei `mat2.dat`.

B.3.2. Daten laden Mit dem Befehl `load A` versucht MATLAB die in `A.mat` gespeicherten Daten in einem Datenfeld `A` zu speichern. Auch ASCII-Dateien kann MATLAB lesen. Da es hier jedoch keine Standardendung gibt, ist die Datei inklusive Endung anzugeben. Es ist darauf zu achten, dass ein rechteckiges Feld an Daten vorliegt, d.h. dass m Zeilen

mit jeweils n numerischen Werten vorliegen. MATLAB erstellt dann eine $m \times n$-Matrix mit dem Namen der Datei ohne Suffix.

```
≫ load  mat1.dat
≫ whos
    Name         Size     Elements  Bytes  Density  Complex
    mat1        3 by 5          15    120     Full    No
 Grand total is 15 elements using 120 bytes
```

B.4. Ausgabe von Text

Mit dem Befehl `fprintf` lassen sich Strings, d.h. Zeichenfolgen, auf dem Bildschirm ausgeben.

```
≫ fprintf('\n Hello world %12.3e\n',4);
  Hello world    4.000e+00
```

Man sieht, dass der auszugebende Text zusätzliche Zeichen enthält, die nicht mit ausgedruckt werden. Diese Zeichen nennt man Escape-Sequenzen. In obigem Beispiel ist die Sequenz `\n` eingebaut. `\n` steht für „newline“ und sorgt dafür, dass bei der Textausgabe an diesen Stellen eine neue Zeile begonnen wird. Der Ausdruck `%12.3e` dient als Platzhalter für einen reellen Wert, der durch Komma getrennt hinter der Zeichenkette folgt. Dabei sei die Zahl in Exponentialdarstellung auszugeben, wofür 12 Stellen mit 3 Nachkommastellen bereitgestellt. Im Beispiel ist dies der Wert 4. Anstatt eines expliziten Wertes können auch Variablen oder Ausdrücke, z.B. `3 * 4` stehen. Ein Platzhalter kann mehrfach in einem `printf`-Befehl vorkommen. In diesem Fall müssen hinter der Zeichenkette genau so viele Werte folgen, wie Platzhalter angegeben sind. Die Reihenfolge der Werte muss mit der Reihenfolge der Platzhalter übereinstimmen, da die Ausdrücke von links nach rechts bewertet werden. Die Escape-Sequenzen dürfen im Text an beliebiger Stelle stehen.

Ausgabeformate

Befehl	Ausgabe
fprintf('%.0e\n',1.234567)	1e00+
fprintf('%.2e\n',1.234567)	1.23e00+
fprintf('%.5e\n',1.234567)	1.23456e00+
fprintf('%10.0e\n',1.234567)	1e00+
fprintf('%10.2e\n',1.234567)	1.23e00+
fprintf('%10.5e\n',1.234567)	1.2346e00+
fprintf('%10.2f\n',1.234567)	1.23
fprintf('%10.5f\n',1.234567)	1.23457

MATLAB rundet numerische Werte bei der Ausgabe, wenn nötig!

B.5. Kontrollbefehle

B.5.1. For-Schleifen **1.** Mit jeglicher gültigen Matrix-Darstellung lässt sich eine FOR-Schleife definieren, z.B.

```
≫ data = [1 7 3 2; 5 4 7 2]
data =
     1 7 3 2
     5 4 7 2
≫ for n=data
   x=n(1)-n(2)
  end
x =
     -4
x =
      3
x =
     -4
x =
     0
```

2. FOR-Schleifen können nach Belieben geschachtelt werden.

```
≫ for k=3:5
   for l=4:-1:2
    A(k,l)=k^2-l;
   end
  end
≫ A
A =
      0  0  0  0
      0  0  0  0
      0  7  6  5
      0 14 13 12
      0 23 22 21
```

3. FOR-Schleifen sollten vermieden werden, wann immer sie durch eine äquivalente Matrix- Darstellung ersetzt werden können. Der folgende Ausdruck ist darunten in optimierter Version aufgeführt.

```
≫ for n=1:10
   x(n)=sin(n*pi/10);
  end
≫x
x =
     Columns 1 through 7
       0.3090   0.5878   0.8090   0.9511   1.0000   0.9511   0.8090
     Columns 8 through 10
       0.5878   0.3090   0.0000
```

```
≫ n=1:10;
≫ x=sin(n*pi/10);
≫ x
x =
     Columns 1 through 7
       0.3090   0.5878   0.8090   0.9511   1.0000   0.9511   0.8090
     Columns 8 through 10
       0.5878   0.3090   0.0000
```

4. Um die Ausführungsgeschwindigkeit zu maximieren, sollte benötigter Speicherplatz vor Ausführung der FOR-Schleife alloziert werden.

```
≫ x=zeros(1,10);
≫ x(1)=0.5;
≫ for n=2:10
    x(n)=x(n-1)*sin(n*pi/10);
  end
≫ x
x =
     Columns 1 through 7
       0.5000   0.2939   0.2378   0.2261   0.2261   0.2151   0.1740
     Columns 8 through 10
       0.1023   0.0316   0.0000
```

B.5.2. WHILE-Schleifen

WHILE-Schleifen sind wie folgt aufgebaut.

```
while Aussage
      Anweisungen
end
```

Die Anweisungen zwischen `while` und `end` werden so lange ausgeführt, wie `Aussage` wahr ist, z.B.

```
≫ num=0; EPS=1;
≫ while (1+EPS) > 1
    EPS=EPS/2;
    num=num+1;
  end
≫ num
num =
      53
```

B.5.3. IF-ELSE-END Konstrukte Eine IF-ELSE-END-Schleife enthält nach dem IF eine Aussage, die daraufhin überprüft wird, ob sie wahr oder falsch ist. Ist sie wahr, so werden die in den folgenden Zeilen stehenden Anweisungen ausgeführt und die Schleife beendet. Ist sie falsch, so erfolgen die Anweisungen, die dem ELSE folgen (ELSE ist optional). Solch ein Konstrukt kann erweitert werden um beliebig viele ELSIF Befehle, die dieselbe Funktion haben wie der am Anfang stehende IF Befehl, aber nur beachtet werden, falls alle vorher überprüften Aussagen falsch sind.

```
if Aussage1
      Anweisungen, wenn Aussage1 wahr
elseif Aussage2
      Anweisungen, wenn Aussage1 falsch und Aussage2 wahr
else
      Anweisungen, wenn Aussage1 und Aussage2 falsch
end
```

B.5.4. Relationen und logische Operatoren Einige Relationen und logische Operatoren sind in den folgenden Tabellen gelistet.

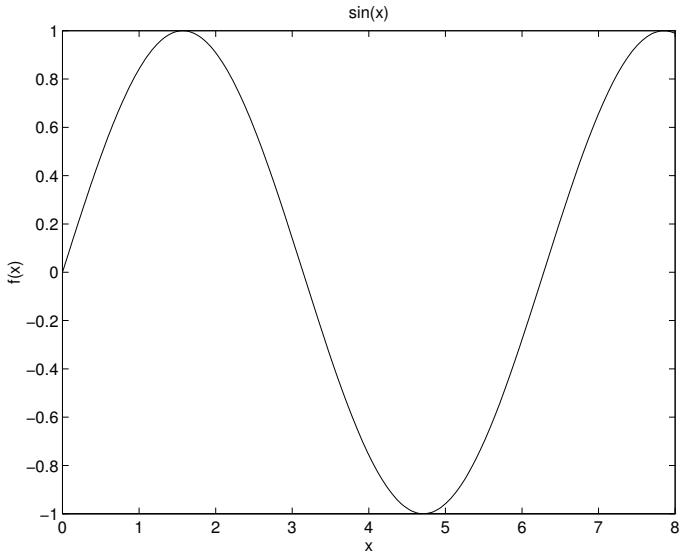
Relationen	
<	kleiner als
<=	kleiner als oder gleich
>	größer als
>=	größer als oder gleich
==	gleich
~=	ungleich

logische Operatoren	
&	UND
|	ODER
~	NICHT

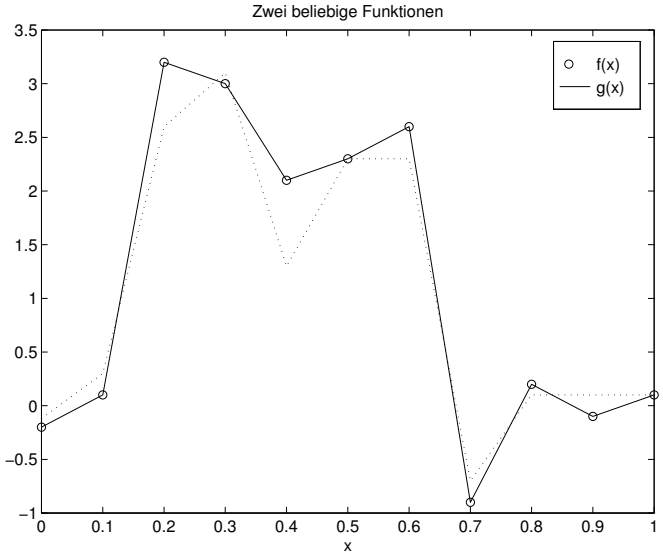
B.6. Graphische Darstellung

B.6.1. Zweidimensionale Graphiken

```
≫ f='sin(x)';
≫ fplot(f,[0 8]);
≫ title(f),xlabel('x'),ylabel('f(x)');
```




```
≫ x=0:0.1:1;
≫ y=[-0.2 0.1  3.2  3  2.1  2.3  2.6 -0.9 0.2 -.1 .1];
≫ z=[-0.12 0.3  2.6  3.1  1.3  2.3  2.3 -0.7 0.1 .1 .1];
≫ plot(x,y,'o',x,y,x,z,':');
≫ title('Zwei beliebige Funktionen'),xlabel('x');
≫ legend('f(x)','g(x)')
```



Linientypen und Farben

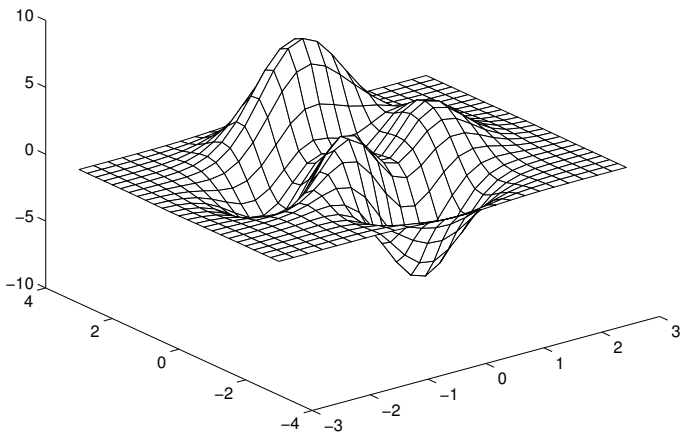
Symbol	Farbe	Symbol	Linientyp
y	gelb	·	Punkt
m	magenta	∘	Kreis
c	cyan	x	x-Markierung
r	rot	+	+-Markierung
g	grün	∗	Sternchen
b	blau	−	durchgezogene Linie
w	weiß	:	gepunktete Linie
k	schwarz	−.	Strichpunkt-Linie
		−−	gestrichelte Linie

2-D Graphikanweisung

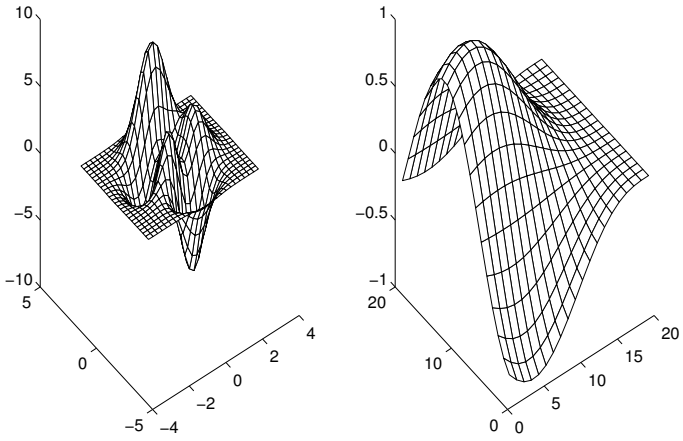
axis	modifiziert die Axen-Proportionen
clf	löscht die Graphik im Graphik-Fenster
close	schließt das Graphik-Fenster
grid	erzeugt ein achsenparalleles Gitter
hold	ermöglicht das Überlagern von Graphiken
subplot	erstellt mehrere Teilgraphiken in einem Fenster
text	gibt Text an vorgegebener Stelle aus
title	zeigt einen Titel an
xlabel	beschriftet die x-Achse
ylabel	beschriftet die y-Achse
colormap(white)	wechselt die Farbtabelle, für S/W-Monitore

B.6.2. Dreidimensionale Graphiken

```
≫ [X,Y,Z] = peaks(25)
≫ mesh(X,Y,Z)
```



```
≫ subplot(1,2,1);
≫ [X,Y,Z] = peaks(25);
≫ mesh(X,Y,Z);
≫ subplot(1,2,2);
≫ X=1:20;
≫ Y=1:20;
≫ Z(X,Y)=(-(cos(X/4)))'*(sin((20-Y)/10).^3);
≫ mesh(X,Y,Z);
```



B.6.3. Graphiken drucken

Graphik-Druckbefehl		
print [-dAusgabetyp] [-Optionen] [Dateiname]		
Ausgabetyp	-dps	Postscript für Schwarzweißdrucker
	-dpsc	Postscript für Farbdrucker
	-deps	Encapsulated Postcript
	-depsc	Encapsulated Color Postcript
Optionen	-P<Drucker>	Spezifiziert den zu benutzenden Drucker

Die Eingabe

```
≫ print fig4 -deps
```

erzeugt die Datei `fig4.eps`.

B.7. Fortgeschrittenes

B.7.1. MATLAB-Skripte Wie schon in Abschnitt B.1 zu Skript-Dateien erwähnt, lässt sich eine Abfolge von MATLAB-Befehlen auch in einer Datei speichern. Diese kann man

dann am Befehl-Fenster aufrufen und die gespeicherte Folge von Befehlen wird ausgeführt. Solche Dateien werden als MATLAB-Skripte oder M-Files bezeichnet. Alle o.g. Befehle lassen sich in einer Datei z.B. mit dem Namen `abc.m` zusammenstellen. Für die Wahl des Dateinamens gelten dabei die folgenden Regeln:

- das erste Zeichen ist ein Buchstabe und
- die Datei hat die Endung **.m** .

Um ein solches M-File zu erstellen, ruft man den Editor auf, der es erlaubt Text einzugeben und diesen als Datei zu speichern. Wenn man den Editor gestartet hat, gebe man die folgenden Befehle ein und speichere die Datei unter dem Namen `abc.m`

```
a = 1;
b = 3;
c = -5;
x1 = (-b + sqrt(b^ 2 - 4*a*c))/(2*a)
x2 = (-b - sqrt(b^ 2 - 4*a*c))/(2*a)
```

Um nun die Befehle aus der Datei `abc.m` auszuführen, gibt man im MATLAB-Befehlsfenster

```
≫ abc
```

ein. MATLAB sucht im aktuellen Pfad nach der Datei `abc.m` und führt die darin enthaltenen Befehle aus. Das aktuelle Verzeichnis wird angezeigt, wenn man

```
≫ pwd
```

eingibt (pwd, engl. print working directory).

B.7.2. Erstellen eigener Funktionen Es wird sicherlich etwas komfortabler sein, eine eigene Funktion zu haben, der man a,b und c als Argument übergibt un die beiden Wurzeln als Eregebnis erhält, als jedesmal erneut ein eigenes M-File anzufertigen. Ein solches Programm könnte z.B. die folgende Datei `root.m` sein.

```
%-----------------
modified "abc.m"
%-----------------

a = input('Enter a: ');
b = input('Enter b: ');
c = input('Enter c: ');

x1 = (-b + sqrt(b^ 2 - 4*a*c))/(2*a)
x2 = (-b - sqrt(b^ 2 - 4*a*c))/(2*a)
```

Die Prozentzeichen in den erstn Zeilen dienen der Dokumentation. Alles was einem solchen Zeichen folgt, wird von MATLAB nicht ausgewertet, sprich interpretiert. Die Argumente werden in deieser Funktion jedoch nur bedingt übergeben und ausgegeben. Eine Funktion, die diese Aufgabe erfüllt, ist die folgende.

```
%-----------------
modified "abc.m"
%-----------------

function [x1, x2] = quadroot(a,b,c)

radical = sqrt(b^2 - 4*a*c);

x1 = (-b + radical)/(2*a)
x2 = (-b - radical)/(2*a)
```

Wenn eine Funktion keine Rückgabewerte hat, können die eckigen Klammern mit den Variablen und das Gleichheitszeichen fehlen. Fehlen die Eingabeparameter, so kann auch der Klammerausdruck nach `quadroot` fehlen. Auf die in der Funktion verwendeten Variablen, hier `radical`, kann man vom Befehlsfenster nicht zugreifen. Ist die Funktion in der Datei `quadroot.m` gespeichert, so erhält man die Wurzeln, indem man

```
[x1, x2] = quadroot(1,3,-5);
```

eingibt. Es kann vom Befehlsfenster oder einer anderen Datei immer nur die erste Funktion in einer Datei aufgerufen werden. Dies bedeutet, in einer Datei können mehrere Funktionen stehen, aber nur die erste Funktion kann extern aufgerufen werden. Alle weiteren Funktionen können nur von Funktionen in der gleichen Datei aufgerufen werden. Für den Anfang ist es einfacher, wenn jede Datei nur eine Funktion enthält. Entscheidend für den Funktionsnamen ist der Name der Datei.

Literatur

[1] Wolfgang Arendt und Karsten Urban. *Partielle Differenzialgleichungen: Eine Einführung in analytische und numerische Methoden*. Spektrum Akademischer Verlag, 2010.

[2] Peter Deuflhard und Andreas Hohmann. *Numerische Mathematik. 1*. Fourth. de Gruyter Lehrbuch. [de Gruyter Textbook]. Eine algorithmisch orientierte Einführung. [An algorithmically oriented introduction]. Walter de Gruyter & Co., Berlin, 2008, S. xii+375. ISBN: 978-3-11-020354-7.

[3] Gerd Fischer. *Lineare Algebra*. Fifth. Bd. 17. Grundkurs Mathematik [Foundational Course in Mathematics]. In collaboration with Richard Schimpl. Friedr. Vieweg & Sohn, Braunschweig, 1979, S. vi+248. ISBN: 3-528-17217-7.

[4] Roland Freund und Ronald Hoppe. *Stoer/Bulirsch: Numerische Mathematik 1*. 10. Aufl. Springer, 2007.

[5] Gene H. Golub und Charles F. Van Loan. *Matrix computations*. Fourth. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, Baltimore, MD, 2013, S. xiv+756. ISBN: 978-1-4214-0794-4; 1-4214-0794-9; 978-1-4214-0859-0.

[6] Wolfgang Hackbusch. *Iterative solution of large sparse systems of equations*. Second. Bd. 95. Applied Mathematical Sciences. Springer, [Cham], 2016, S. xxiii+509. ISBN: 978-3-319-28481-1; 978-3-319-28483-5. DOI: 10.1007/978-3-319-28483-5. URL: https://doi.org/10.1007/978-3-319-28483-5.

[7] Günther Hämmerlin und Karl-Heinz Hoffmann. *Numerische Mathematik*. Fourth. Springer-Lehrbuch. [Springer Textbook]. Grundwissen Mathematik. [Basic Knowledge in Mathematics]. Springer-Verlag, Berlin, 1994, S. xiv+449. ISBN: 3-540-58033-6. DOI: 10.1007/978-3-642-57894-6. URL: https://doi.org/10.1007/978-3-642-57894-6.

[8] Martin Hanke-Bourgeois. *Grundlagen der numerischen Mathematik und des wissenschaftlichen Rechnens*. Third. Vieweg + Teubner, Wiesbaden, 2009, S. 840. ISBN: 978-3-8348-0708-3. DOI: 10.1007/978-3-8348-9309-3. URL: https://doi.org/10.1007/978-3-8348-9309-3.

[9] Martin Hermann. *Numerische Mathematik*. expanded. Oldenbourg Verlag, Munich, 2006, S. xii+522. ISBN: 978-3-486-57935-2; 3-486-57935-5. DOI: 10.1524/9783486595055. URL: https://doi.org/10.1524/9783486595055.

[10] Martin Hermann. *Numerische Mathematik*. expanded. Oldenbourg Verlag, Munich, 2011, S. xiv+563. ISBN: 978-3-486-70820-2. DOI: 10.1524/9783486708202. URL: https://doi.org/10.1524/9783486708202.

[11] Donald E. Knuth. *The art of computer programming. Vol. 4A. Combinatorial algorithms. Part 1.* Addison-Wesley, Upper Saddle River, NJ, 2011, S. xv+883. ISBN: 978-0-201-03804-0; 0-201-03804-8.

[12] Andreas Meister. *Numerik linearer Gleichungssysteme.* Eine Einführung in moderne Verfahren. [An introduction to modern procedures]. Friedr. Vieweg & Sohn, Braunschweig, 1999, S. x+222. ISBN: 3-528-03135-2. DOI: 10.1007/978-3-322-93899-2. URL: https://doi.org/10.1007/978-3-322-93899-2.

[13] Wilhelm Niethammer. »Relaxation bei nichtsymmetrischen Matrizen«. In: *Math. Z.* 85 (1964), S. 319–327. ISSN: 0025-5874. DOI: 10.1007/BF01110678. URL: https://doi.org/10.1007/BF01110678.

[14] Kaspar Nipp und Daniel Stoffer. *Lineare Algebra. Eine Einführung für Ingenieure unter besonderer Berücksichtigung numerischer Aspekte.* vdf Hochschulverlag ETH Zürich, 2001. ISBN: 372812818X.

[15] Gerhard Opfer. *Numerische Mathematik für Anfänger.* expanded. Grundkurs Mathematik. [Foundational Course in Mathematics]. Eine Einführung für Mathematiker, Ingenieure und Informatiker. [An introduction for mathematicians, engineers and computer scientists]. Vieweg + Teubner, Wiesbaden, 2008, S. xx+392. ISBN: 978-3-8348-0413-6.

[16] Les Piegl und Wayne Tiller. *The NURBS Book.* second. New York, NY, USA: Springer-Verlag, 1996.

[17] Robert Plato. *Numerische Mathematik kompakt.* Second. Grundlagenwissen für Studium und Praxis. [Foundations for study and practice]. Friedr. Vieweg & Sohn, Wiesbaden, 2004, S. xvi+414. ISBN: 3-528-13153-5. DOI: 10.1007/978-3-322-93922-7. URL: https://doi.org/10.1007/978-3-322-93922-7.

[18] Alfio Quarteroni, Riccardo Sacco und Fausto Saleri. *Numerical mathematics.* Second. Bd. 37. Texts in Applied Mathematics. Springer-Verlag, Berlin, 2007, S. xviii+655. ISBN: 978-3-540-34658-6; 3-540-34658-9. DOI: 10.1007/b98885. URL: https://doi.org/10.1007/b98885.

[19] Alfio Quarteroni, Fausto Saleri und Paola Gervasio. *Scientific computing with MATLAB and Octave.* Bd. 2. Texts in Computational Science and Engineering. Fourth edition [of MR2253397]. Springer, Heidelberg, 2014, S. xviii+450. ISBN: 978-3-642-45366-3; 978-3-642-45367-0. DOI: 10.1007/978-3-642-45367-0. URL: https://doi.org/10.1007/978-3-642-45367-0.

[20] H. R. Schwarz. *Numerik symmetrischer Matrizen.* Unter Mitwirkung von H. Rutishauser und E. Stiefel. Leitfäden der Angewandten Mathematik und Mechanik, Band 11. B. G. Teubner, Stuttgart, 1968, S. 243.

[21] Hans Rudolf Schwarz. *Numerische Mathematik.* Fourth. With a contribution by Jörg Waldvogel. B. G. Teubner, Stuttgart, 1997, S. 653. ISBN: 3-519-32960-3. DOI: 10.1007/978-3-663-01227-6. URL: https://doi.org/10.1007/978-3-663-01227-6.

[22] G. W. Stewart. »On the sensitivity of the eigenvalue problem $Ax = \lambda Bx$«. In: *SIAM J. Numer. Anal.* 9 (1972), S. 669–686. ISSN: 0036-1429. DOI: 10.1137/0709056. URL: https://doi.org/10.1137/0709056.

[23] Josef Stoer. *Numerische Mathematik. 1.* Seventh. Springer-Lehrbuch. [Springer Textbook]. Eine Einführung—unter Berücksichtigung von Vorlesungen von F. L. Bauer. [An introduction—based on the lectures of F. L. Bauer]. Springer-Verlag, Berlin, 1994, S. xii+367. ISBN: 3-540-57823-4. DOI: 10.1007/978-3-662-09023-7. URL: https://doi.org/10.1007/978-3-662-09023-7.

[24] Josef Stoer und Roland Bulirsch. *Numerische Mathematik. 2.* Third. Springer-Lehrbuch. [Springer Textbook]. Eine Einführung—unter Berücksichtigung von Vorlesungen von F. L. Bauer. [An introduction, with reference to lectures by F. L. Bauer]. Springer-Verlag, Berlin, 1990, S. xiv+341. ISBN: 3-540-51482-1. DOI: 10.1007/978-3-662-22250-8. URL: https://doi.org/10.1007/978-3-662-22250-8.

[25] J. H. Wilkinson. *The algebraic eigenvalue problem.* Monographs on Numerical Analysis. Oxford Science Publications. The Clarendon Press, Oxford University Press, New York, 1988, S. xviii+662. ISBN: 0-19-853418-3.

Stichwortverzeichnis

L

M

N

O

P

Q

R

S

T

U

V

Z