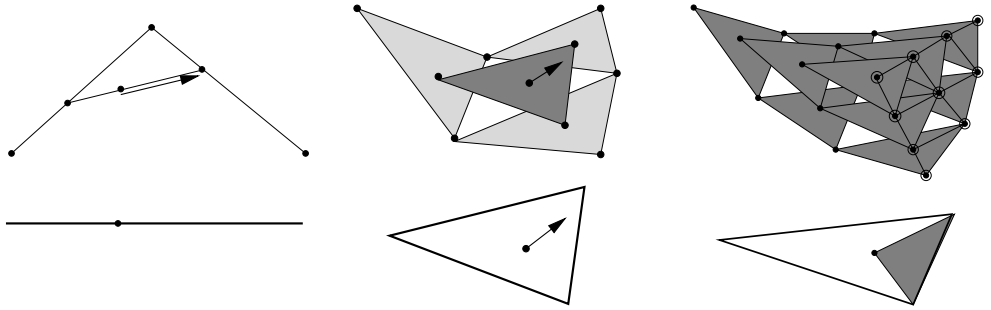
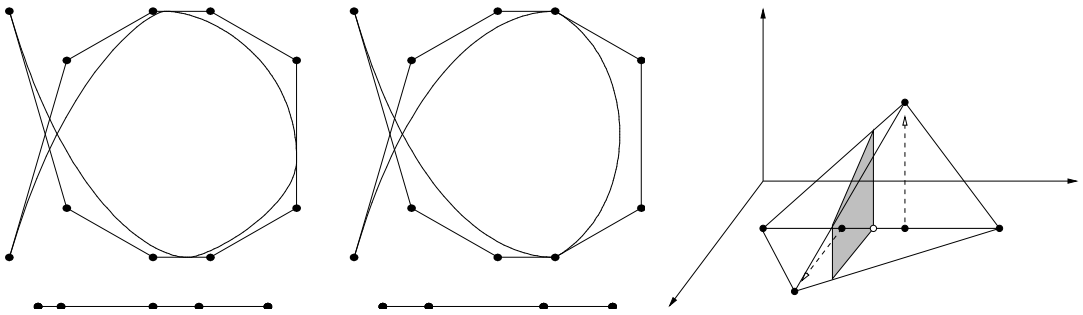


Splinekurven und -flächen in CAGD und Anwendung

Vorlesung, zuerst gehalten im Sommersemester 2012

Tomas Sauer

Version 0.0
Letzte Änderung: 31.3.2012



Statt einer Leerseite . . . | 0

Welcher aber . . . durch die Geometria sein Ding beweist und die gründliche Wahrheit anzeigt, dem soll alle Welt glauben. Denn da ist man gefangen.

Albrecht Dürer

A peculiarity of the higher arithmetic is the great difficulty which has often been experienced in proving simple general theorems which had been suggested quite naturally by numerical evidence.

H. Davenport, *The Higher Arithmetic*, 1952

I always found it shameful that mere technologists should have arrogated to themselves the right to be called that, scientists, men of knowledge.

S. Rushdie, *Grimus*

You may call it 'nonsense' if you like, [. . .] but I've heard nonsense, compared with which that would be as sensible as a dictionary.

L. Carroll, *Through the looking glass*

Tomas Sauer
Lehrstuhl für Mathematik
mit Schwerpunkt Digitale Bildverarbeitung
Universität Passau
Innstr. 43
94032 Passau

0 Inhaltsverzeichnis

Geschichte muß doch wohl allein auf Treu und Glauben angenommen werden? Nicht?

Lessing, Nathan der Weise III,7

1 Eine kurze Geschichte der Splines

Auch wenn man gerne die Arbeiten (Schoenberg, 1949a; Schoenberg, 1949b) von Schoenberg aus dem Jahr 1946 als die "Geburtsstunde" der Splines bezeichnet, waren die B–Splines "wahrscheinlich Hermite und sicherlich Peano" (Schoenberg in (Schoenberg, 1973)) bekannt. Will man noch weiter in der Geschichte zurückgehen, so kann man sich auf die *exponential Euler Splines* berufen, die bereits von Euler (1755), aber natürlich nicht unter diesem Namen, untersucht wurden. Eine Zusammenfassung der "Ur- und Frühgeschichte" der Splines gibt beispielsweise die Arbeit von Butzer, Schmidt und Stark (Butzer *et al.*, 1988).

Doch kehren wir wieder zurück zu Isaac J. Schoenberg, einem der profiliertesten und vielseitigsten angewandten und nicht nur angewandten Mathematiker des 20. Jahrhunderts. Im Rahmen seiner Tätigkeit im War Department at Aberdeen Proving Ground während des zweiten Weltkriegs begann er, ausgehend von Approximationsproblemen, mit der Untersuchung von Splines. Es ist einer der wenigen positiven Aspekte des Kriegs, daß zur selben Zeit auch Haskell B. Curry, ansonsten eher der Algebra und abstrakten Logik zugetan, von diesem Ansatz Kenntnis bekam und ihn gemeinsam mit Schoenberg weiterentwickelte. Eine Kopie eines Briefes, in dem Curry von diesen Entwicklungen erzählt, findet sich im Buch von Micchelli (Micchelli, 1995) (das sich beispielsweise mit den Themen Subdivision, Splines mit verallgemeinerten Differenzierbarkeitseigenschaften, multivariate Splines und Blossoming befaßt und schon deswegen sehr empfehlenswert ist). Trotzdem beschäftigte sich auch Schoenberg zuerst einmal lange Zeit mit anderen Dingen, so daß die berühmte Arbeit von Curry und Schoenberg (Curry & Schoenberg, 1966) erst mit fast 20–jähriger Verspätung im Jahre 1966 erschien. Einer der Gründe, warum Schoenberg seine Resultate über Splines wieder "aus der Schublade" holte, war wohl die Tatsache, daß inzwischen auch andere Mathematiker, allen voran Carl de Boor, die Splines entdeckt hatten und so Schoenbergs Interesse neu weckten. Natürlich bestand einer der Gründe für das aufkeimende allgemeine Interesse an Splines im Bedarf an "gu-

ten" Typen von Kurven im Zusammenhang mit den CAD–Systemen und den numerischen Berechnungen, die durch die Entwicklung der Computertechnik in ganz neuen Größenordnungen durchgeführt wurden. Auf der anderen Seite waren und sind Splines auch ein wichtiges theoretisches Hilfsmittel bei der Untersuchung von n–*ten Weiten*, wo sie eine wichtige Rolle als Minimallösungen bestimmter Funktionale spielen. Inzwischen ist die Literatur zum Thema "Splines" gigantisch. Die beste, umfangreichste und via Internet zugängliche Bibliographie stammt von de Boor und kann beispielsweise über seine WWW–Homepage `http://www.cs.wisc.edu/~deboor` abgerufen werden.

In den meisten "klassischen" Lehrbüchern über Numerische Mathematik werden Splines (oft nur kubisch, das heißt von der Ordnung $m = 3$) als Lösung eines Interpolationsproblems an den *einfachen Knoten* $t_{m+1}, \ldots, t_n$ eingeführt und beschrieben. Nicht, daß das irgend etwas schlechtes wäre, gerade in diesem Zusammenhang zeigen die Splines ein den Polynomen weit überlegenes Verhalten, da sie das oftmals katastrophale Oszillationsverhalten der Interpolationspolynome nicht mitmachen. Auf alle Fälle hat man es aber in diesem Rahmen wegen der einfachen Knoten stets mit Funktionen aus C^{m-1} zu tun. Eine Basis des zugehörigen Splineraums ist dann schnell angegeben: sie besteht aus allen Polynomen vom Grad m sowie den *abgebrochenen Potenzen*

$$(t - t_i)_+^m = \begin{cases} (t - t_i)^m & t \geq t_i, \\ 0 & t < t_i, \end{cases} \quad i = m+1, \ldots, n-1.$$

Klar – diese Funktionen sind alle linear unabhängig, stückweise polynomial und $m - 1$ mal stetig differenzierbar. Damit haben wir also $m + 1$ Polynome und $n - m - 1$ abgebrochene Potenzen, also n Basisfunktionen, und da wir, dank der B–Splines, wissen, daß der Splineraum Dimension n hat, sind diese Funktionen ebenfalls eine Basis. Allerdings eine Basis mit Nachteilen: die Basisfunktionen haben unendlichen Träger! Deswegen war man interessiert an einer Basis des Splineraums dergestalt, daß alle Basisfunktionen *möglichst kleinen* Träger haben – und das führte zu den B–Splines. Daß diese Funktionen dann auch noch über eine Rekursionsformel verknüpft sind, wie Cox (Cox, 1972) und de Boor (Boor, 1972) zeigten, macht sie nur noch sympathischer. Zudem haben die Basisfunktionen mit kompaktem Träger noch einen weiteren Vorteil, den bereits Schoenberg weidlich ausnutzte: da zur Berechnung von $\mathbf{N}^m(t)$ stets nur die m Knoten "links von t" und die m Knoten "rechts von t", sowie die $m + 1$ zugehörigen Kontrollpunkte verwendet werden, gibt es überhaupt keine Schwierigkeiten, auch *unendliche* Knotenfolgen t_i, $i \in \mathbb{Z}$, mit unendlichen Kontrollpolygonen $\mathbf{d}_i$, $i \in \mathbb{Z}$, zu betrachten. Und das ist mit einer Basis aus Polynomen und abgebrochenen Potenzen nun doch etwas unschön, da hier an jedem Punkt eine *unendliche* Anzahl von Basisfunktionen beiträgt. Aber besser noch: sind die Knoten *gleichverteilt*, also z.B. $t_i = i$, $i \in \mathbb{Z}$, (das sind genau die

Cardinal B–Splines von Schoenberg (Schoenberg, 1973)), dann sind die B–Splines lediglich verschobene Kopien voneinander, genauer

$$N_i^m(x) = N_0^m(x - i), \qquad i \in \mathbb{Z},$$

und man hat eigentlich nur *einen* B–Spline.

Aus den 70er und 80er Jahren gibt es eine Vielzahl von Arbeiten über Splines, die für einige Zeit ein richtiges "Modethema" waren. Einen gewissen Überblick aus unterschiedlichen Perspektiven und von unterschiedlichen Zugängen her bieten beispielsweise die Bücher von de Boor (Boor, 1978), Schumaker (Schumaker, 1981) oder Nürnberger (Nürnberger, 1989). Auf alle Fälle fand man eine Vielzahl von Problemen und Gebieten, wo sich die Splines als mehr oder weniger nützlich erwiesen.

Trotzdem geschah noch einmal das Wunder, daß auf einem eigentlich "abgegrasten" Feld, das die (univariaten) Splines in den 80er Jahren mit Sicherheit waren, noch einmal zarte Blüten (engl. "*blossoms*") sprossen. Es war überraschenderweise wieder de Casteljau, der zuerst auf die Zusammenhänge mit polaren Formen aufmerksam machte (Casteljau, 1985), wenn auch in sehr schwer lesbarer Form. Die Anwendung von Blossoming auf Splinekurven geht wohl auf Lyle Ramshaw (Ramshaw, 1987) zurück (siehe auch und vor allem (Ramshaw, 1989)). Die Darstellung in diesem Skript folgt in weiten Zügen (Sauer, 1996), die ihrerseits auf einer Arbeit von Hans–Peter Seidel (Seidel, 1989) basiert, welche im wesentlichen für meine Begeisterung für dieses Gebiet verantwortlich ist. Seidels Ziel war es übrigens, die neuen Einsichten, die das Blossoming gewährte, auch auf Flächen (generell, auf den multivariaten Fall) zu übertragen. Obwohl da einige Dinge ganz entschieden anders sind als bei den Kurven, gelang es doch, Splineflächen von hochgradiger Flexibilität zu konstruieren. Aber das ist eine andere Geschicht ...

Ich schwöre Ihnen, jedes Wort ist wahr. Für Sätze aber hafte ich nicht.

R. Schami

2 Kurven, Funktionen und so weiter

Die Vorlesung beschäftigt sich mit der Modellierung, Speicherung und Manipulation von Kurven und Flächen[1]. Damit wir auch wissen wovon wir reden und um sicherzustellen, daß wir auch wirklich von demselben reden, ist es sinnvoll, zuerst einmal die grundlegenden Begriffe zu wiederholen bzw. einzuführen. So eine Art „Differentialgeometrie für Arme“ also.

2.1 Kurven

Eine **Kurve** im $\mathbb{R}^d$, $d \geq 1$, kann auf viele verschiedene Arten eingeführt werden, beispielsweise implizit, parametrisch oder approximativ. Auch wenn wir uns im Rahmen der Vorlesung auf parametrische Kurven beschränken wollen, sollen die beiden anderen nicht ganz unerwähnt bleiben.

Eine **implizite Kurve** ist die Lösung eines Gleichungssystems der Form

$$F(\mathbf{x}) = 0, \qquad F = (f_1, \ldots, f_n), \qquad \mathbf{x} \in \mathbb{R}^d.$$

In vielen Fällen ist dabei $n = d - 1$, getreu dem Motto „jede Bedingung bindet eine Dimension“, das aber natürlich für nichtlineare Funktionen F blanker Unsinn und auch für lineares F mit ewtas Vorsicht zu geniessen ist. In der Praxis sind implizite Kurven ausgesprochen schwer zu handhaben[2], da für jeden Punkt der Kurve irgendwie ein Gleichungssystem zu lösen ist. Und Informationen wie Glattheit der Kurve, Tangenten in einem Punkt und ähnliches sind noch schwerer zu erhalten.

Eine **approximative Kurve** kennt in vielen Fällen noch nicht einmal eine explizite Darstellung der Kurve oder Fläche, sondern verfügt nur über Approximationen, die normalerweise von sehr einfacher Natur sind, beispielsweise

[1] Auch wenn das Hauptaugenmerk auf den Kurven liegen wird.

[2] Das stimmt natürlich so wieder einmal nicht ganz, beim Raytracing ist es beispielsweise besser, die zu rendernden Flächen implizit gegeben zu haben, da der Schnitt solcher Flächen mit einem Sehstrahl „nur“ das Hinzufügen einer weiteren Gleichung ist.

Linienzüge, die sich aber wenn nötig beliebig verfeinern lassen, bis sie der Kurve so nahe kommen wie man nur will. Subdivision–Kurven und –Flächen, die beispielsweise bei der 3D–Animation zum Einsatz kommen[3] sind Beispiele hierfür. Geeignet sind solche Darstellungen vor allem dann, wenn das Objekt nur gut aussehen muss und analytische Eigenschaften nicht wirklich relevant sind.

Eine **parametrische Kurve** ist eine Abbildung

$$\mathbf{f} = \begin{bmatrix} f_1 \\ \vdots \\ f_d \end{bmatrix} : I \to \mathbb{R}^d, \qquad I \subset \mathbb{R} \text{ kompakt.} \tag{2.1}$$

Man spricht von einer stetigen bzw. k–mal stetig differenzierbaren Kurve[4], wenn $f_j \in C(I)$ bzw. $f_j \in C^k(I)$, $j = 1, \ldots, d$. Nur weil eine Kurve glatt im Sinne von Differenzierbarkeit ist, muss sie übrigens noch lange nicht glatt im optischen Sinne sein. Ein einfaches Beispiel ist die **Neilsche Parabel**, siehe Abb 2.1, der

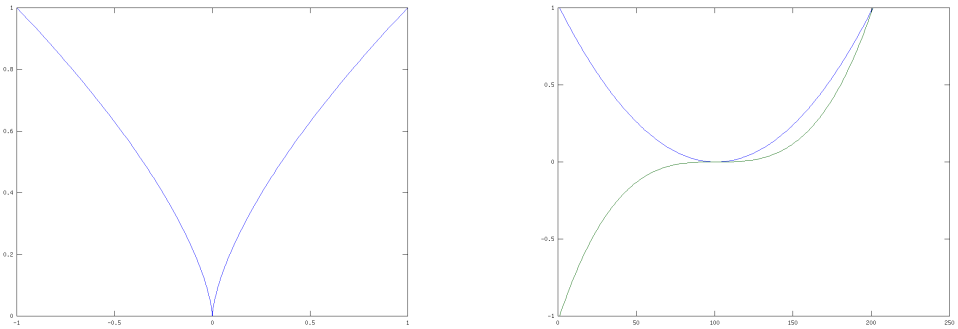


Abbildung 2.1: Die Neilsche Parabel *(links)* und ihre Komponenten *(rechts)*. Man sieht deutlich, daß die Komponentenfunktionen glatt sind, die Parabel aber natürlich eine Spitze in der Mitte hat.

einfachen Kurve

$$\mathbf{f}(u) = \begin{bmatrix} u^3 \\ u^2 \end{bmatrix}, \qquad I = [-1, 1],$$

[3] Pixar stellt angeblich zumindest sei *„Gerri's Game"* alle Figuren mit Subdivisionflächen dar, die sehr intuitiv zu manipulieren sind.

[4] Der Begriff „glatt" wird gerne auch mal für eine unspezifizierte Differenzierbarkeitsordnung verwendet.

die zwar zu $C^\infty(I)^d$ gehört[5], aber dennoch eine „Zacke“ an $tu = 0$ hat. Mit etwas genauem Hinschauen sieht man auch den Grund, denn dort ist $\dot{\mathbf{f}}(0) := \frac{df}{du}(0) = \mathbf{0}$.

Definition 2.1 *Eine Stelle* $u \in I$ *heißt* ***kritischer Punkt*** *der Kurve* $\mathbf{f}$ *falls* $\dot{\mathbf{f}}(u) = \mathbf{0}$ *ist.*

Kritische Punkte einer Kurve sind etwas sehr unangenehmes und sollten im Normalfall vermieden oder zumindest sorgfältig untersucht werden, denn:

> An einem kritischen Punkt kann fast alles passieren (muss aber nicht).

Da im Rahmen der Vorlesung ja auch `Matlab` bzw. `octave` verwendet werden sollen, ist es vielleicht ganz nett und instruktiv, sich anzusehen, wie man so einen Plot wie in Abb. 2.1 hinbekommt. Zuerst beschafft man sich eine Abtastung des Intervalls I,

```
octave> X = (-1:.01:1);
```

und dann plottet man die beiden Funktionen:

```
octave> plot( X.^3,X.^2 );
```

Will man die Komponenten geplottet haben, so geht das mit

```
octave> plot( [ X.^3,X.^2 ]');
```

Das `plot`–Kommando ist etwas gewöhnungsbedürftig und benimmt sich halt unterschiedlich, je nachdem, ob man eine Matrix oder zwei Vektoren übergibt.

2.2 Graph oder Funktion oder was?

Der nächste Punkt über den man sich bei Kurven klar werden muss, ist, ob man sie als Graphen, also als *Punktmenge* $\mathbf{f}(I)$ oder wirklich als Funktionen, also als *Abbildung* $\mathbf{f} : I \to \mathbb{R}^d$ ansieht. Während ein Designer sich im wesentlichen für $\mathbf{f}(I)$ interessiert, ist bei Bahnplanungsaufgaben, beispielsweise im Bereich des *Numerical Control*, spielt hingegen die Funktion eine zentrale Rolle, denn da geht es dann auch um sowas wie Ableitungen. Der wesentliche Unterschied zwischen diesen beiden Sichtweisen wird ersichtlich, wenn man sich Reparametrisierungen ansieht.

[5] Polynome sind unendlich oft differenzierbar.

Definition 2.2 (Reparametrisierung) *Eine **Reparametrisierung** ist eine* surjektive *Funktion* $\varphi : J \to I$ *von einem kompakten Intervall* J *mit der Eigenschaft*[6] $\varphi(J) = I$. *Eine **reguläre Reparametrisierung** erfüllt ausserdem* $\varphi \in C^1(J)$ *und* $\dot{\varphi} \neq 0$.

Klar, dem Graphen sind Reparametrisierungen völlig egal, denn da gilt ja

$$\mathbf{f}(I) = \mathbf{f}(\varphi(J)) = (\mathbf{f} \circ \varphi)(J) =: \mathbf{f}_\varphi(J),$$

wobei wir $\mathbf{f}_\varphi := \mathbf{f} \circ \varphi$ für die **reparametrisierte Kurve** schreiben werden. Für die Funktion macht das natürlich einen Riesenunterschied. Nehmen wir beispielsweise

$$I = J = [0,1], \qquad \varphi(u) = \begin{cases} 2u, & 0 \le u \le \frac{1}{2}, \\ 1-2u, & \frac{1}{2} \le u \le 1, \end{cases} \tag{2.2}$$

so durchläuft $\mathbf{f}_\phi$ die Kurve gleich zweimal und das ist, wie man sich leicht überlegen kann, keine Frage der (Nicht-)Differenzierbarkeit von φ. Immerhin: Bei einer *regulären* Reparametrisierung ist nicht nur der Mehrfachdurchlauf ausgeschlossen, sondern man kann diese Parametrisierung auch wieder umkehren.

Übung 2.1 Zeigen Sie: Ist $\dot{\varphi} \neq 0$, so existiert die Umkehrabbildung $\varphi^{-1} : I \to J$. ◇

Kritischen Punkten ist die Reparametrisierung übrigens egal, ganz egal, ob diese regulär ist oder nicht: Ist $\varphi(u)$ nämlich ein kritischer Punkt von $\mathbf{f}$ und $\varphi \in C^1(J)$, so gilt

$$\frac{d}{du}\mathbf{f}_\varphi(u) = \frac{d}{du}\mathbf{f}(\varphi(u)) = \underbrace{\dot{\mathbf{f}}(\varphi(u))}_{=\mathbf{0}}\,\dot{\varphi}(u) = \mathbf{0},$$

und deswegen ist u ein kritischer Punkt von $\mathbf{f}_\varphi$. Ist φ regulär, dann besteht sogar eine $1-1$–Beziehung zwischen den kritischen Punkten, „schlechte“ Reparametrisierungen mit Nullstellen der Ableitung können sogar noch weitere kritische Punkte generieren.

Wie das Beispiel (2.2) zeigt, kann man mit schlechten Reparametrisierungen eine ganze Menge erreichen, so daß sich ein Kompromiss anbietet: Eine **intrinsische Eigenschaft** einer Kurve ist eine, die zumindest unter *regulären* Reparametrisierungen invariant bleibt[7]. Das wichtigste Hilfsmittel dafür ist die **Bogenlänge**. Dazu schreiben wir $I = [u_a, u_e]$ für unser Parameterintervall und approximieren die Kurve durch einen Streckenzug durch die Punkte

$$\mathbf{f}(u_j), \quad j = 0, \ldots, n, \qquad u_a = u_0 < u_1 < \cdots < u_n =: u^* \le u_e,$$

siehe Abb. 2.2. Mit der euklidischen Norm

[6] Das ist gerade die Surjektivität.

[7] Mehr können wir nicht fordern, Eigenschaften, die unter *allen* Reparametrisierungen invariant bleiben, sind ziemlich trivial.

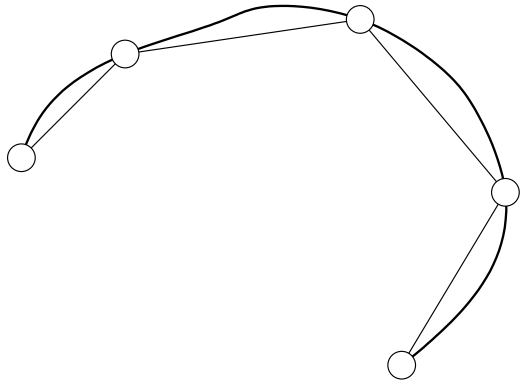


Abbildung 2.2: Die Näherung der Bogenlänge über den interpolierenden Streckenzug.

$$\|\mathbf{x}\|_2 = \sqrt{\sum_{j=1}^{d} x_j^2}$$

und unter Verwendung der **Taylor–Formel** nullter Ordnung[8]

$$\mathbf{f}(u') - \mathbf{f}(u) = (u' - u)\ \dot{\mathbf{f}}(\xi), \qquad \xi \in [u, u']$$

ergibt sich die Länge des Streckenzugs als

$$\ell\,(\mathbf{f}; u_0, \ldots, u_n) = \sum_{j=1}^{n} \left\|\mathbf{f}\,(u_j) - \mathbf{f}\,(u_{j-1})\right\|_2 = \sum_{j=1}^{n} (u_j - u_{j-1})\ \left\|\dot{\mathbf{f}}\,(\xi_j)\right\|_2,$$

was nun auch wieder nichts anderes als die **Riemann–Summe** für das Integral

$$\ell\,(\mathbf{f}; u^*) = \int_{u_a}^{u^*} \left\|\dot{\mathbf{f}}(u)\right\|_2 \, du \tag{2.3}$$

ist.

Definition 2.3 (Bogenlänge) *Die **Bogenlänge** $s(u^*)$ des Kurvenstücks von u_e bis u^* ist der durch (2.3) definierte Wert $\ell\,(\mathbf{f}; u^*)$.*

[8] Die Intervallschreibweise in dieser Gleichung ist im Sinne konvexer Hüllen zu verstehen, d.h. $[u, u'] = [\min\{u, u'\}, \max\{u, u'\}]$.

Die Bogenlänge ist invertierbar, solange es keine Intervalle gibt, auf denen $\dot{\mathbf{f}}$ verschwindet. Solche Intervalle sind zwar nicht auszuschliessen, aber eigentlich idiotisch, denn sie legen eine Parametrisierung zugrunde, bei der $\mathbf{f}$ „auf der Stelle steht“ und man verliert nichts, wenn man solche Intervalle zu einem, dann kritischen, Punkt zusammenfasst. Außerdem hängt die Bogenlänge nicht von der (regulären) Parametrisierung ab:

$$\begin{aligned}\ell\left(\mathbf{f}_\varphi; \varphi^{-1}(u)\right) &= \int_{\varphi^{-1}(u_a)}^{\varphi^{-1}(u_e)} \left\| \frac{d}{du}\mathbf{f}(\varphi(v)) \right\|_2 dv = \int_{\varphi^{-1}(u_a)}^{\varphi^{-1}(u_e)} \left\| \dot{\mathbf{f}}(\varphi(v))\ \dot{\varphi}(v) \right\|_2 dv \\ &= \int_{\varphi^{-1}(u_a)}^{\varphi^{-1}(u_e)} \left\| \dot{\mathbf{f}}(\varphi(v)) \right\|_2 |\dot{\varphi}(v)|\ dv = \int_{u_a}^{u} \left\| \dot{\mathbf{f}}(v) \right\|_2 dv \\ &= \ell(\mathbf{f}; u)\,.\end{aligned}$$

Ist nun aber die Funktion $u \mapsto s(u)$ invertierbar[9], so können wir die Kurve $\mathbf{f}$ auch nach der Bogenlänge parametrisieren und erhalten so $\mathbf{f}(s)$, für die insbesondere die Identität[10]

$$\begin{aligned}\dot{\mathbf{f}}(u) &:= \frac{d}{du}\mathbf{f}(s(u)) = \frac{d}{ds}\mathbf{f}(s(u))\ \frac{d}{du}s(u) \\ &= \mathbf{f}'(s(u))\ \frac{d}{du}\int_{u_a}^{u} \left\| \dot{\mathbf{f}}(v) \right\|_2 dv = \mathbf{f}'(s(u))\ \left\| \dot{\mathbf{f}}(u) \right\|_2\end{aligned}$$

gilt, was wir auch als[11]

$$\mathbf{f}'(s) = \frac{\dot{\mathbf{f}}(u(s))}{\left\| \dot{\mathbf{f}}(u(s)) \right\|_2} \tag{2.4}$$

schreiben können, weswegen die Ableitung nach der Bogenlänge immer $\left\| \mathbf{f}'(s) \right\|_2 = 1$ erfüllt, also eine **Einheitstangente** liefert.

2.3 Ebene Kurven vs. Raumkurven

Einen weiteren großen Unterschied macht es, ob man Kurven in der Ebene $\mathbb{R}^s$ oder im Raum $\mathbb{R}^3$ betrachtet. Das ist bei Geraden schon anschaulich klar: In der Ebene schneiden sich Geraden fast immer, im Raum fast nie. Und über das, was im $\mathbb{R}^d$ für $d > 3$ passiert, wollen wir uns gar nicht erst Gedanken machen, zumal das nicht wirklich praxisrelevante Situationen sind. Natürlich ist das hier

[9] Also praktisch immer!

[10] Normalerweise bezeichnet man in diesem Kontext mit $\dot{\mathbf{f}}$ die Ableitung nach dem (freien) Bahnparameter, mit $\mathbf{f}'$ hingegen die Ableitung nach der Bogenlänge.

[11] Diese Gleichung ist an kritischen Punkten allerdings ein wenig problematisch! Noch ein Grund, sie zu meiden.

nur eine „Mini–Übersicht“, detaillierte Informationen gibt's beispielsweise in (Struik, 1961).

Betrachten wir also mal so ein $\mathbf{f} : I \to \mathbb{R}^d$, $d = 2, 3$, und bilden an der Stelle $u \in I$ wieder einmal eine Taylor–Entwicklung, dann ist für $\delta \in \mathbb{R}$,

$$\mathbf{f}(u+\delta) = \mathbf{f}(u) + \delta\,\dot{\mathbf{f}}(u) + \frac{\delta^2}{2}\ddot{\mathbf{f}}(u) + \frac{\delta^3}{3}\dddot{\mathbf{f}}(u) + \cdots = \sum_{j=0}^{\infty} \frac{\delta^j}{j!}\,\mathbf{f}^{(j)}(u),$$

wobei wir uns die Terme höherer Ordnung schenken wollen. Als nächstes fordern wir, daß die Ableitungsvektoren $\mathbf{f}^{(j)}$, $j = 1, \ldots, d$, linear unabhängig sind, dann bilden sie ein lokales Koordinatensystem, das man nach Gram–Schmidt orthonormalisieren kann:

$$\begin{aligned}
\mathbf{t} &:= \frac{\dot{\mathbf{f}}(u)}{\left\|\dot{\mathbf{f}}(u)\right\|_2},\\
\widetilde{\mathbf{n}} &:= \ddot{\mathbf{f}}(u) - \mathbf{t}\,\mathbf{t}^T\ddot{\mathbf{f}}(u) = \ddot{\mathbf{f}}(u) - \frac{\dot{\mathbf{f}}^T(u)\ddot{\mathbf{f}}(u)}{\left\|\dot{\mathbf{f}}(u)\right\|_2^2}\dot{\mathbf{f}}(u)\\
&= \frac{\ddot{\mathbf{f}}(u)\,\dot{\mathbf{f}}^T(u)\dot{\mathbf{f}}(u) - \dot{\mathbf{f}}(u)\,\ddot{\mathbf{f}}^T(u)\dot{\mathbf{f}}(u)}{\left\|\dot{\mathbf{f}}(u)\right\|_2^2} = \frac{(\mathbf{F}(u) - \mathbf{F}^T(u))\,\dot{\mathbf{f}}(u)}{\left\|\dot{\mathbf{f}}(u)\right\|_2^2}, \qquad \mathbf{F} := \ddot{\mathbf{f}}\dot{\mathbf{f}}^T,\\
\mathbf{n} &:= \frac{\widetilde{\mathbf{n}}}{\left\|\widetilde{\mathbf{n}}\right\|_2} = \frac{\dot{\mathbf{f}}^T(u)\dot{\mathbf{f}}(u)\,\ddot{\mathbf{f}}(u) - \dot{\mathbf{f}}^T(u)\ddot{\mathbf{f}}(u)\,\dot{\mathbf{f}}(u)}{\left\|\dot{\mathbf{f}}^T(u)\dot{\mathbf{f}}(u)\,\ddot{\mathbf{f}}(u) - \dot{\mathbf{f}}^T(u)\ddot{\mathbf{f}}(u)\,\dot{\mathbf{f}}(u)\right\|_2} = \frac{(\mathbf{F}(u) - \mathbf{F}^T(u))\,\dot{\mathbf{f}}(u)}{\left\|(\mathbf{F}(u) - \mathbf{F}^T(u))\,\dot{\mathbf{f}}(u)\right\|_2},\\
\mathbf{b} &:= \frac{\dot{\mathbf{f}}(u) \times \ddot{\mathbf{f}}(u)}{\left\|\dot{\mathbf{f}}(u) \times \ddot{\mathbf{f}}(u)\right\|_2}.
\end{aligned}$$

Dabei verwenden wir im $\mathbb{R}^3$ das (vektorwertige) **Kreuzprodukt**, das als[12]

$$\mathbf{x} \times \mathbf{y} = \det\begin{bmatrix} \mathbf{e}_1 & x_1 & y_1 \\ \mathbf{e}_2 & x_2 & y_2 \\ \mathbf{e}_3 & x_3 & y_3 \end{bmatrix} = \begin{bmatrix} x_2y_3 - x_3y_2 \\ x_3y_1 - x_1y_3 \\ x_1y_2 - x_2y_1 \end{bmatrix}$$

definiert ist. Die **Tangente t** kennen wir schon, die ist rein zufällig $\mathbf{f}'(s)$, $s = s(u)$ und wegen

$$1 = \left\|\mathbf{f}'(s)\right\|_2^2 = (\mathbf{f}'(s))^T\,\mathbf{f}'(s) \qquad \Rightarrow \qquad 0 = \frac{d}{ds}\left((\mathbf{f}'(s))^T\,\mathbf{f}'(s)\right) 2\,(\mathbf{f}'(s))^T\,\mathbf{f}''(s)$$

muss die **Normale n** ein Vielfaches von $\mathbf{f}''(s)$ sein, schließlich liegt

$$\frac{d^2}{ds^2}\mathbf{f}(u(s)) = \frac{d}{ds}\left(\dot{\mathbf{f}}(u(s))\;u'(s)\right) = \ddot{\mathbf{f}}(u(s))\;(u'(s))^2 + \dot{\mathbf{f}}(u(s))\;u''(s) \tag{2.5}$$

[12] Eigentlich kennt man Determinanten ja nur für quadratische Matrizen, aber mit der folgenden formal nicht korrekten Formel kann man sich's am besten merken.

im Erzeugnis von $\dot{\mathbf{f}}$ und $\ddot{\mathbf{f}}$ und steht senkrecht auf $\mathbf{t}$, also auch auf $\dot{\mathbf{f}}$. Die **Binormale** $\mathbf{b}$ existiert nur für $d = 3$ und ist gerade dadurch definiert, daß sie auf $\dot{\mathbf{f}}$ und $\ddot{\mathbf{f}}$ senkrecht steht – das ist ja die Natur und Rechtfertigung des Kreuzprodukts.

Übung 2.2 Zeigen Sie, daß sich im $\mathbb{R}^3$ die Normale auch durch

$$\mathbf{n} = \mathbf{t} \times \mathbf{b} = \frac{\dot{\mathbf{f}} \times \left(\dot{\mathbf{f}} \times \ddot{\mathbf{f}}\right)}{\left\|\dot{\mathbf{f}}\right\|_2 \left\|\dot{\mathbf{f}} \times \ddot{\mathbf{f}}\right\|_2}$$

berechnen lässt. ◊

Definition 2.4 (Begrifflichkeiten)

1. *Die Vektoren* $\mathbf{t}, \mathbf{n}, \mathbf{b}$ *bilden das* ***Frenet–Dreibein****, englisch* ***Frenet Frame****, an der Stelle* u*, eine natürliches und intrinsisches lokales Koordinatensystem.*

2. *Die* ***Schmiegeebene***[13] *bzw.* ***Osculating Plane*** *ist die von* $\mathbf{t}$ *und* $\mathbf{n}$ *aufgespannte* affine *Ebene durch* $\mathbf{f}(u)$ *im* $\mathbb{R}^3$*. Im* $\mathbb{R}^2$ *ist sie bedeutungslos.*

3. *Die* ***Krümmung*** *der Kurve ist als*

$$\kappa(u) = \left\|\widetilde{\mathbf{n}}\right\|_2 = \left\|\mathbf{f}''(u)\right\|_2$$

definiert und gibt gleichzeitig den Reziprokwert des Radius des Schmiegekreises[14] *an die Kurve an der Stelle* u *an.*

Die Auswirkungen der unterschiedlichen Parametrisierungen auf Krümmung und so weiter kann man sich sehr einfach an einem ganz elementaren Beispiel veranschaulichen.

Beispiel 2.5 *Für* $I = [0, 1]$ *betrachten wir das Geradenstück*

$$\mathbf{f}(u) = \mathbf{a}\,u + \mathbf{b}, \qquad \mathbf{0} \neq \mathbf{a}, \mathbf{b} \in \mathbb{R}^d, \qquad d = 2, 3. \tag{2.6}$$

Solche Geraden haben Tangeten $\mathbf{a}/\|\mathbf{a}\|_2$ *und natürlich*[15] *Krümmung Null, was wir hier auch durch die Ableitung gezeigt bekommen:*

$$\dot{\mathbf{f}}(u) = \mathbf{a}, \qquad \ddot{\mathbf{f}}(u) = \mathbf{0}.$$

[13] Der Name stammt von (einem der) Johann Bernoulli(s), oder, wie es in (Struik, 1961) heisst, von „John Bernoulli".

[14] Was auch immer das ist.

[15] Ja, das ist anschaulich klar, aber gerade in der Mathematik ist das, was anschaulich klar ist, selten richtig. Dennoch sollte man sich hier im klaren sein, daß jeder vernünftige Krümmungsbegriff so formuliert sein muss, daß Geraden Krümmung Null haben – denn Krümmung ist ja gerade ein Maß für die Abweichung von der Geraden. Definition sind nie richtig oder falsch, aber sie können sinnvoll oder idiotisch sein.

Nun parametrisieren wir um, und zwar mit der regulären[16] *Parametrisierung* $\varphi(u) = u^2$*, so daß wir* $\mathbf{f}_\varphi(u) = \mathbf{a}u^2 + \mathbf{b}$*, also*

$$\dot{\mathbf{f}}_\varphi = 2\mathbf{a}u, \qquad \ddot{\mathbf{f}}_\varphi = 2\mathbf{a} \neq \mathbf{0}$$

erhalten. Dennoch ist die Krümmung Null, denn $\dot{\mathbf{f}}_\varphi$ *und* $\ddot{\mathbf{f}}_\varphi$ *sind ja* linear abhängig *und die Normale als Krümmungsrichtung ist ja der Teil von* $\ddot{\mathbf{f}}_\varphi$*, der auf* $\dot{\mathbf{f}}_\varphi$ *senkrecht steht.*

Übung 2.3 Was ist die Bogenlängenparametrisierung von $\mathbf{f}$ aus (2.6)? ◊

Man sieht: Parametrisierung nach der Bogenlänge ist für die Behandlung der *Geometrie* von Kurven sehr hilfreich. Kann man nun aber auch jede Kurve nach der Bogenlänge parametrisieren? Theoretisch ja, praktisch ist aber $\ell(\mathbf{f}; \cdot)$ eine ziemlich komplizierte Funktion, die man nur schwer invertieren kann. Einfach wäre das nur, wenn beispielsweise $\left\|\dot{\mathbf{f}}\right\|_2$ ein Polynom wäre, das liesse sich dann exakt integrieren und invertieren. Solche Kurven bezeichnet man als **Hodographen**, aber sie sind ausgesprochen spezielle Kurven, bei denen ja die Komponenten voneinander abhängen. Allerdings sind es auch besonders „schöne“ und nützliche Kurven, wie man in (Farouki, 2007) nachlesen kann.

Bemerkung 2.6 (Parametrisierungen) *Im Numerical Control werden Kurven oftmals in verschiedenen Stufen des Prozesses in unterschiedlichen Formen gespeichert. Die* Bahngeometrie *wird gerne nach der Bogenlänge parametrisiert*[17]*, für die* Bahndynamik *hingegen verwendet man zumeist eine Zeitparametrisierung, denn dann sind die Ableitungen von* $\mathbf{f}$ *nach dem Zeitparameter* t *die physikalischen Größen* ***Geschwindigkeit****,* ***Beschleunigung*** *und* ***Ruck***[18]*, die auf einer Maschine bestimmte Grenzwerte nicht überschreiten dürfen.*

2.4 Anschlussbedingungen

In der Praxis sind Kurven (und Flächen) zumeist stückweise definiert, wobei die Objekte auf jedem der Stücke „brav“, also von einer bestimmten Glattheit im Sinne von Differenzierbarkeit sind. An den Verbindungsstellen wird es aber natürlich interessanter. Nehmen wir also an, wir hätten zwei Kurven, $\mathbf{f} : [u_0, u^*]$ und $\mathbf{g} : [u^*, u_1]$, deren Übergang wir an der Stelle u^* ansehen wollen.

Stetigkeit ist eine Bedingung, über die wir nicht zu reden brauchen. Wenn resultierende zusammengesetzte Kurve $[\mathbf{f}, \mathbf{g}]$ an u^* nicht einmal stetig ist, dann

[16] Die eine Nullstelle der Ableitung an $u = 0$ ist irrelevant, wer's nicht glaubt kann $\varphi(u) = (1-\varepsilon)u^2 + \varepsilon u$ für $0 < \varepsilon < 1$ verwenden und ein bisschen mehr rechnen.

[17] Zumindest approximativ.

[18] Terminus Technicus für die Beschleunigungswechsel.

sind es zwei separate Kurven und fertig. Wir fordern also immer bei einem Übergang, daß

$$\mathbf{f}(u^*) = \mathbf{g}(u^*) \tag{2.7}$$

sein soll.

Differenzierbarkeit wird da schon interessanter. Da wäre zuerst einmal die **parametrische Differenzierbarkeit** C^1, bei der wir *zusätzlich*

$$\dot{\mathbf{f}}(u^*) = \dot{\mathbf{g}}(u^*) \tag{2.8}$$

verlangen, was allerdings wegen $\dot{\mathbf{f}}_\varphi = \dot{\mathbf{f}}\,\dot{\varphi}$ nicht unter Reparametrisierungen erhalten bleibt. Eine intrinsische Variante ist die **geometrische Differenzierbarkeit** G^1, bei der nur noch

$$\mathbf{f}'(u) = \frac{\dot{\mathbf{f}}(u)}{\left\|\dot{\mathbf{f}}(u)\right\|_2} = \frac{\dot{\mathbf{g}}(u)}{\left\|\dot{\mathbf{g}}(u)\right\|_2} = \mathbf{g}'(u) \tag{2.9}$$

gefordert wird. G^1 hat damit drei äquivalente Beschreibungen:

1. Die Einheitstangenten stimmen überein bzw. die Tangenten zeigen in dieselbe Richtung.

2. Man kann die Kurve $\mathbf{f}$ regulär so reparametrisieren, daß $\mathbf{f}_\varphi$ und $\mathbf{g}$ einen C^1–Übergang haben. Dazu muss nur gewährleistet werden, daß[19], daß

$$\varphi(u) = u \qquad \text{und} \qquad \dot{\varphi}(u) = \frac{\left\|\dot{\mathbf{g}}(u)\right\|_2}{\left\|\dot{\mathbf{f}}(u)\right\|_2}$$

   ist.

3. Parametrisiert man beide Kurven nach der Bogenlänge, so erhält man einen C^1–Übergang. Das ist nun wieder eine intrinische, geometrische Eigenschaft.

Man sieht also, es gibt parametrische und parameterfreie Beschreibungen der geometrischen Differenzierbarkeit. Natürlich ist jede C^1–Verbindung auch G^1, in letzterem Fall hat man ja einen Freiheitsgrad mehr und kann die Länge von $\dot{\mathbf{f}}(u)$ frei wählen, solange sie positiv[20] bleibt.

[19] Und das wird nur bei kritischen Punkten problematisch – hatte ich schon gesagt, daß man diese nach Kräften vermeiden sollte?

[20] Negative Länge würde eine Tangente bedeuten, die in die Gegenrichtung zeigt und das will man dann doch nicht.

Noch interessanter wird es, wenn man sich die **zweimalige Differenzierbarkeit** ansieht. Nun gut, parametrisch ist das einfach, da kommt nur

$$\ddot{\mathbf{f}}(u^*) = \ddot{\mathbf{g}}(u^*) \tag{2.10}$$

zu (2.7) und (2.8) dazu, aber wie sieht das geometrisch aus? Fordern wir $\mathbf{f}' = \mathbf{g}'$ und $\mathbf{f}'' = \mathbf{g}''$, so liefert das die bereits bekannte Bedingung (2.9), die wir auch als

$$\mathbf{g}'(u^*) = \dot{\mathbf{f}}(u^*)\, u', \qquad u' = u'(s(u^*)) = \left\|\dot{\mathbf{f}}(u^*)\right\|_2^{-1}, \tag{2.11}$$

schreiben können und zu der ausserdem noch

$$\mathbf{g}''(u^*) = \ddot{\mathbf{f}}(u^*)\,(u')^2 + \dot{\mathbf{f}}(u^*)\, u'' \tag{2.12}$$

dazukommt. Das heisst, daß $\mathbf{g}''$ in der Schmiegeebene von $\mathbf{f}$ an der Stelle u liegen muss bzw. die beiden Schmiegeebenen übereinstimmen müssen[21], da $\dot{\mathbf{f}}$ und $\dot{\mathbf{g}}$ kollinear sind und daß derselbe Wert von u' wie in (2.11) auch hier funktionieren muss, allerdings quadratisch. (2.11) und (2.12) sind insgesamt $2d$ Gleichungen in den beiden Unbekannten u' und u'', von denen aber die Hälfte nichtlinear ist. So einfach geht es also offensichtlich nicht. Deswegen verwendet man für G^2 gerne die folgende Definition, siehe (Farin, 1988).

Definition 2.7 (G^2) *Zwei parametrische Kurven $\mathbf{f}$ und $\mathbf{g}$ an u haben einen G^2–Übergang, wenn sie an u einen C^1–Übergang haben[22] und in der Bogenlängenparametrisierung C^2 aneinanderhängen.*

Diese Begrifflichkeiten machen das Leben einfacher, denn nun ist nach (2.11)

$$u'_{\mathbf{f}} = \left\|\dot{\mathbf{f}}(u^*)\right\|_2^{-1} = \left\|\dot{\mathbf{g}}(u^*)\right\|_2^{-1} = u'_{\mathbf{g}} =: u',$$

und es gilt nach (2.12), daß

$$\begin{aligned} \mathbf{0} &= f''(u^*) - g''(u^*) = \ddot{\mathbf{f}}(u^*)\,(u')^2 + \dot{\mathbf{f}}(u^*)\, u''_{\mathbf{f}} - \ddot{\mathbf{g}}(u^*)\,(u')^2 + \dot{\mathbf{g}}(u^*)\, u''_{\mathbf{g}} \\ &= (u')^2 \left(\ddot{\mathbf{f}}(u^*) - \ddot{\mathbf{g}}(u^*) + \dot{\mathbf{f}}(u^*)\, \frac{u''_{\mathbf{f}} - u''_{\mathbf{g}}}{(u')^2} \right). \end{aligned}$$

Das können wir abschliessend auch mal als Satz formulieren.

Satz 2.8 *Ist u^* kein kritischer Punkt, dann ist ein C^1–Übergang genau dann ein G^2–Übergang, wenn es ein $\lambda \in \mathbb{R}$ gibt, so daß $\ddot{\mathbf{f}}(u^*) - \ddot{\mathbf{g}}(u^*) = \lambda\, \dot{\mathbf{f}}(u^*)$ ist.*

[21] Für $d = 2$ ist das trivial, bei Raumkurven hingegen eine echte Forderung!

[22] In der globalen Parametrisierung auf $[u_0, u_1]$

Infinities and indivisibles transcend our finite understanding, the former on account of their magnitude, the latter because of their smallness; Imagine what they are when combined.

G. Galilei

3 Eine algorithmische Einführung in Splines

Splinekurven spielen auch heute noch eine wesentliche Rolle in industriellen Anwendungen, insbesondere im Bereich der CNC–Maschinen[23]. Das geht so weit, daß die Steuerung derartiger Maschinen, sei es zum Schneiden, zum Fräsen oder zum Drehen, Splines bis zur Ordnung 5 als Eingaben direkt verarbeiten kann. Diese Anwendungen beeinflussen natürlich auch die Sichtweise auf Splines und die Art und Weise, wie man mit ihnen umgeht. Dieses erste Kapitel wird daher von Anwendungen und dem praktischen Umgang mit Splines beeinflusst sein; manche der Beweise werden wir auf spätere Kapitel verlegen, denn eleganter kann man diese Sachen oftmals mit Hilfe von etwas allgemeineren theoretischen Mitteln angehen.

3.1 Zur Motivation

Im praktischen Kontext sind **Splines** stückweise polynomiale Kurven zur *Darstellung* und *Manipulation* von Kurven. Und wenn wir "Kurven" sagen, dann meinen wir Kurven, also nicht Graphen von Funktionen, sondern **parametrische Kurven**, also Abbildungen $\mathbf{f} : [a, b] \to \mathbb{R}^d$, wobei das Parameterintervall $[a, b]$ normalerweise eher irrelevant ist[24] und d gerne die Werte 2 oder 3 annimmt – wir haben es also wieder mit *planaren Kurven* bzw. *Raumkurven* zu tun. Damit so eine Kurve vernünftig auf dem Computer dargestellt und verarbeitet werden kann, sollte sie zwei wesentliche Eigenschaften haben:

[23] CNC = Computer Numerical Control.

[24] Es wird – zumeist aus Mangel an Phantasie – gerne als $[0, 1]$ gewählt. Eine sinnvollere Wahl wäre eine Parametrisierung der Kurve nach der Bogenlänge, dann hätte man es mit dem Intervall $[0, \ell]$ zu tun, wobei ℓ die Länge der Kurve ist.

1. Die Kurve muss vollständig durch eine *endliche* Anzahl von Koeffizienten $\mathbf{d}_1, \ldots, \mathbf{d}_n \in \mathbb{R}^d$ beschrieben werden. Diese Koeffizienten bezeichnet man auch als **Kontrollpunkte**.

2. Die Kontrollpunkte sollten geometrische Eigenschaften der Kurve wiedergeben.

Während die erste Forderung ziemlich offensichtlich ist, wird die zweite Eigenschaft benötigt, um die "Interaktion zwischen Mensch und Maschine" möglich zu machen oder doch zumindest wesentlich zu erleichtern. Gerade wenn das Design oder die Bearbeitung von Kurven aus der Manipulation von Kontrollpunkten besteht, sollte doch ein intuitiver Zusammenhang zwischen diesen Kontrollpunkten und der endgültigen Gestalt der Kurve bestehen.

3.2 Knoten und B–Splines

Das Splines, wie bereits gesagt, *stückweise polynomiale* Funktionen sein werden, ist es sicherlich zuerst einmal vernünftig, diese Stücke auch entprechend festzulegen, was durch das Konzept der Knotenfolge geschieht.

Definition 3.1 *Eine endliche, indizierte Menge* $\mathsf{T} = \mathsf{T}_{m,n} = \{t_1, \ldots, t_{m+n+1}\}$ *heißt* ***Knotenfolge*** *der Ordnung* m *falls*

1. $t_1 \leq \cdots \leq t_{n+m+1}$.

2. $t_j < t_{j+m+1}$.

Bemerkung 3.2 (Nomenklatur)

1. *Es ist immer eine wichtige Entscheidung, wo man bei der Knotenfolge mit der Indizierung beginnt. Normalerweise bevorzuge ich wie in (Sauer, 2000a) eine Indizierung, die mit Null anfängt, allerdings hat diese den wesentlichen Nachteil, daß eine direkte Umsetzung in* `Matlab` *bzw.* `Octave` *dadurch unmöglich wird*[25]*. Da aber zur Anwendung immer auch das numerische Spiel auf dem Rechner gehört, ist hier eine Indizierung vorzuziehen, die mit* 1 *beginnt.*

2. *Bei dem Begriff der* ***Ordnung*** *eines Splines gibt es in der Literatur keinen wirklichen Standard. Man kann damit entweder*[26] *den lokalen Polynomgrad bezeichnen oder aber die lokale Dimension, also eines mehr als der lokale Polynomgrad. Bei der Verwendung von Literatur zu Splines empfiehlt es sich also, zuerst einmal genau nachzusehen, was genau mit "Ordnung" gemeint ist.*

[25] Im Gegensatz zu Programmierung in C/C++.
[26] Wie in diesem Skript.

Offensichtlich ist die Ordnung der Knotenfolge nur für die zweite Bedinung relevant, die besagt, daß die **Vielfachheit** eines Knotens $m+1$ nicht überschreiten darf; die Vielfachheit $\mu = \mu_j$ eines Knotens t_j ist einfach die Zahl, die angibt, wie oft der Knoten innerhalb der Knotenfolge T wiederholt wird:

$$t_1 \le \cdots \le t_{j-1} < \underbrace{t_j = \cdots = t_{j+\mu-1}}_{\mu} < t_{j+\mu} \le \cdots \le t_{m+n+1}. \tag{3.1}$$

Ist $\mu_j = 1$ für $j = 1, \ldots, m+n+1$, dann sagen wir es handle sich bei T um eine **einfache Knotenfolge**. Und schon können wir unsere B–Splines definieren, diesmal über einen ganz einfachen rekursiven Prozess.

Definition 3.3 *Für* $k = 0, \ldots, m$*, sind die* ***B–Splines*** N_j^k, $j = 1, \ldots, n+m-k$, *der Ordnung* k *definiert als*

$$N_j^0(\cdot \,|\, T) = \chi_{[t_j, t_{j+1})}, \tag{3.2}$$

$$N_j^k(\cdot \,|\, T) = \frac{\cdot - t_j}{t_{j+k} - t_j} N_j^{k-1}(\cdot \,|\, T) + \frac{t_{j+k+1} - \cdot}{t_{j+k+1} - t_{j+1}} N_{j+1}^{k-1}(\cdot \,|\, T). \tag{3.3}$$

Aus dieser Definition erhalten wir sehr unmittelbar bereits einige fundamentale Eigenschaften der B–Splines.

Proposition 3.4 *Die B–Splines*

1. *sind nichtnegativ,* $N_j^k \ge 0$,
2. *haben kompakten Träger,* $N_j^k(x \,|\, T) = 0$, $x \notin [t_j, t_{j+k+1}]$,
3. *sind stückweise Polynome vom Höchstgrad* k *auf den Knotenintervallen:* $N_j^k\big|_{(t_\ell, t_{\ell+1})} \in \Pi_k$.

Beweis: Der Beweis ist eine einfache Induktion über k. Die stückweise konstanten Funktionen N_j^0 haben die obigen drei Eigenschaften beinahe trivialerweise. Für den Übergang $k-1 \to k$ verwenden wir die Rekursionsformel (3.3), die uns zusammen mit der Induktionsannahme sagt, daß

$$N_j^k(x \,|\, T) = 0, \qquad x \notin ([t_j, t_{j+k}] \cup [t_{j+1}, t_{j+k+1}]) = [t_j, t_{j+k+1}]$$

und da für alle $x \in [t_j, t_{j+k+1}]$ alle Terme in (3.3) ≥ 0 sind, muss dies auch für N_j^k gelten. Außerdem ergibt sich N_j^k dadurch, daß zwei Splines der Ordnung $k-1$,

also zwei stückweise lineare Polynome vom Grad $\leq k-1$, mit einer linearen[27] Funktion multipliziert, was wiederum eine stückweise polynomiale Funktion vom Grad $\leq k$ liefert. □

Bemerkung 3.5 *Für* einfache Knoten *funktioniert die rekursive Definition 3.3 der B–Splines glatt und ohne Probleme, aber bei mehrfachen Knoten muss man ein wenig aufpassen, weil man sonst irgendwann bei der Iteration an eine "division by zero" gerät, nämlich wenn $t_{j+k+1} = t_j$ ist. Glücklicherweise kann man diese Situation aber einfach ignorieren, denn in diesem Fall ist der Träger des entsprechenden Splines in der Rekursionsformel die leere Menge und deswegen lässt man ihn einfach unter den Tisch fallen.*

Die Auswertung eines B–Splines an einem Vektor `X` von Punkten wird in `Octave` von einer Funktion[28] `BSplEval` erledigt, also nutzen wir diese Funktion einmal und plotten den einen oder anderen B–Spline, einfach, um ein Gefühl dafür zu bekommen. Ein Blick auf Definition 3.3 zeigt, daß wir für n Funktionen $N_1^m, \ldots, N_n^m$ einen Knotenvektor mit $n+m+1$ Knoten brauchen. Gut, versuchen wir uns doch einmal am "klassischsten" alle Fälle, am kubischen Fall $m = 3$ und verwenden wir einfach equidistante Knoten

```
octave> T = (0:10)
T =

   0   1   2   3   4   5   6   7   8   9  10
```

dann haben wir es mit $n = 11 - 4 = 7$ B–Splines zu tun, die wir an einem feinen Gitter auswerten und dort plotten wollen:

```
octave> hold on; X = (0:.01:10);
octave> for j=1:7 plot( X,BSplEval( j,3,T,X ) ); end
```

Das war ja einfach, also versuchen wir auch noch eine nicht–äquidistante Knotenfolge:

```
octave> T = sqrt(0:10); clearplot; X = (0:.01:max(T));
octave> for j=1:7 plot( X,BSplEval( j,3,T,X ) ); end
```

[27] Die Sprechweise ist nicht immer eindeutig, denn man kann streng genommen ja zwischen linearen und affinen Funktionen unterscheiden, je nachdem, ob noch ein konstanter Term vorhanden ist oder nicht. Trotzdem soll hier "linear" zumeist für Polynome vom Höchstgrad 1 stehen, ganz egal, ob die Dinger nun linear oder affin sind.

[28] Alle Funktionen sind so gestaltet, daß sie auch unter `Matlab` funktionieren sollten, aber da `Octave` Open Source Software ist und somit nicht nur ausgesprochen leistungsfähig sondern von allem auch frei verwendbar, allgemein verfügbar und ohne Lizenzeinschränkungen nutzbar ist, werde ich in Zukunft nur auf `Octave` verweisen.

Die Ergebnisse dieses kleinen Spielchens können in Abb. 3.1, bewundert werden, aber sie können natürlich nur als Motivation für eigene, unabhängige Experimente mit B–Splines dienen, bei denen man ein Gefühl dafür bekommen kann, wie sich die B–Splines in Abhängigkeit von der Knotenwahl verhalten. Jetzt

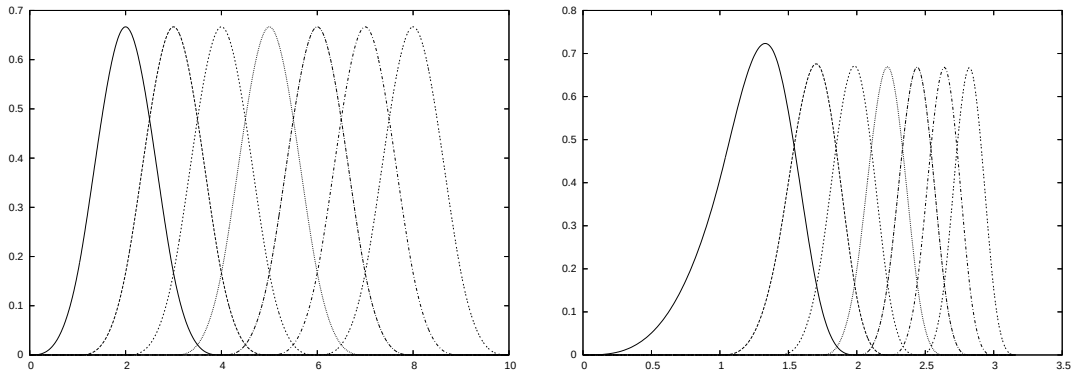


Abbildung 3.1: Die B–splines mit gleichverteilten *(links)* und "Wurzel"–Knoten *(rechts)*.

aber trotzdem zurück zur Theorie!

Definition 3.6 *Eine **Splinekurve** in* $\mathbb{R}^d$ *mit **Kontrollpolygon*** $\mathbf{d} = [\mathbf{d}_j : j = 1, \ldots, n] \in \mathbb{R}^{d \times n}$ *ist eine Kurve der Form*

$$S_m\mathbf{d} = S_m(\cdot \mid T)\,\mathbf{d} := \sum_{j=1}^{n} \mathbf{d}_j\, N_j^m(\cdot \mid T)\,. \tag{3.4}$$

Eine Splinekurve wird also eindeutig bestimmt durch ihre **Kontrollpunkte** $\mathbf{d}_1, \ldots, \mathbf{d}_n$ **control points** $\mathbf{d}_j$, $j = 1, \ldots, n$, und die Knotenfolge T. Als einfache und unmittelbare Konsequenz daraus kann man derartige Kurven vollständig dadurch beschreiben und auch manipulieren, daß man das Kontrollpolygon und/oder die Knotenfolge verändert. Anders gesagt: Alles, was wir mit Splines so anstellen werden und auch können, lässt sich immer auf die Bestimmung geeigneter Knoten und Kontrollpunkte zurückführen. Und die sind *endliche* Information!

3.3 Der Algorithmus von de Boor

Ohne dieses geometrische Verfahren zur Auswertung von Splines geht es natürlich nicht, ganz abgesehen davon, daß er uns auch ganz allgemein beim Verständnis der Natur der Splines unterstützen wird.

Algorithmus 3.7 (de Boor)
Eingabe:

- *Knotenfolge* $\mathsf{T} = \mathsf{T}_{m,n}$,
- *Kontrollpunkte* $\mathbf{d}_1, \ldots, \mathbf{d}_n \in \mathbb{R}^d$,
- $x \in [t_{m+1}, \ldots, t_{n+1}]$.

Prozedur:

1. *Bestimme* $\ell \in \{m+1, \ldots, n\}$ *so daß* $x \in [t_\ell, t_{\ell+1})$.
2. *Initialisiere:*
$$\mathbf{d}_j^0(x) = \mathbf{d}_j, \qquad j = \ell - m, \ldots, \ell.$$
3. *Für* $k = 1, \ldots, m$
   (a) *Für* $j = i - m + k, \ldots, i$ *berechne*[29]
$$\alpha(x) = \alpha_j(x) = \frac{t_{j+m-k+1} - x}{t_{j+m-k+1} - t_j}$$
   und damit
$$\mathbf{d}_j^k(x) = \alpha(x)\, \mathbf{d}_{j-1}^{k-1}(x) + (1 - \alpha(x))\ \mathbf{d}_j^{k-1}(x). \tag{3.5}$$

Ergebnis: $S_m\mathbf{d}(x) = \mathbf{d}_\ell^m(x)$.

Übung 3.1 Implementieren Sie den Algorithmus von de Boor in `Matlab/octave`. ◊

[29] Diese *baryzentrische Koordinate* hängt natürlich von j ab - aber als Variable in einem Programm brauchen wir's nicht.

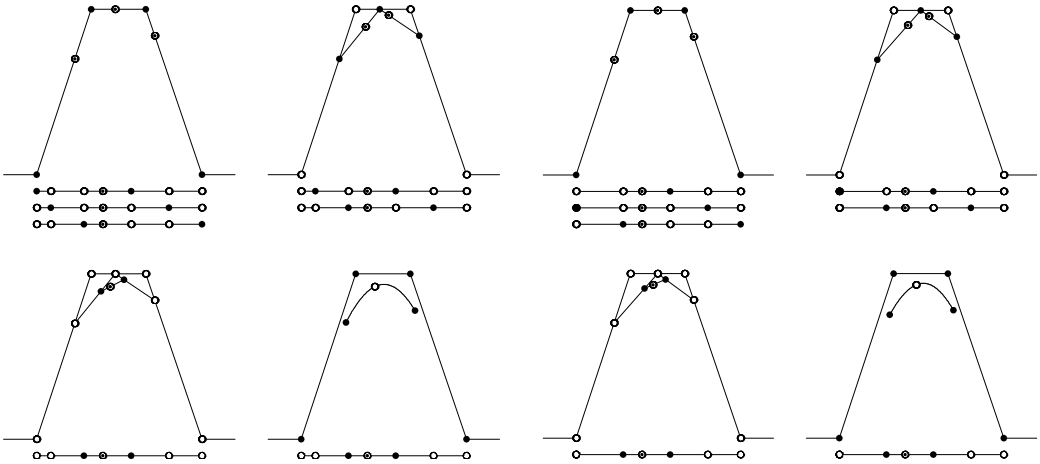


Abbildung 3.2: Der kubische ($m = 3$) de–Boor–Algorithmus um einen einfachen *(links)* und einen doppelten *(rechts)* Knoten herum.

Die Korrektheit von Algorithmus 3.7 lässt sich nun sehr einfach zeigen: Wir wenden einfach die Rekursionsformel für die B–Splines "rückwärts" an:

$$\begin{aligned}
S_m\mathbf{d}(x) &= \sum_{j=1}^{n} \mathbf{d}_j \, N_j^m(x \mid T) \\
&= \sum_{j=1}^{n} \mathbf{d}_j \left[\frac{x - t_j}{t_{j+k} - t_j} \, N_j^{m-1}(x \mid T) + \frac{t_{j+k+1} - x}{t_{j+k+1} - t_{j+1}} \, N_{j+1}^{m-1}(x \mid T) \right] \\
&= \sum_{j=1}^{n} \mathbf{d}_j \frac{x - t_j}{t_{j+k} - t_j} \, N_j^{m-1}(x \mid T) + \sum_{j=2}^{n+1} \mathbf{d}_{j-1} \frac{t_{j+k} - x}{t_{j+k} - t_j} \, N_j^{m-1}(x \mid T) \\
&= \sum_{j=1}^{n+1} \left[\frac{t_{j+k} - x}{t_{j+k} - t_j} \, \mathbf{d}_{j-1} + \frac{x - t_j}{t_{j+k} - t_j} \, \mathbf{d}_j \right] N_j^{m-1}(x \mid T) \\
&= \sum_{j=1}^{n+1} \mathbf{d}_j^1(x) \, N_j^{m-1}(x \mid T) \, .
\end{aligned}$$

Jetzt müssen wir nur noch aufpassen, welche Splines wirklich zur Summe beitragen – da die Trägerintervalle der Splines ja mit fallender Ordnung immer kleiner werden, liegt der Punkt x in immer weniger Trägerintervallen und am Schluss sind fast alle "Zwischenkoeffizienten" und deren Berechnungen unnötig bis auf die, die gerade zu $\mathbf{d}_\ell^m$ führen, und das sind gerade die, die auch im de–Boor–Algorithmus auftauchen.

Bemerkung 3.8 (Weitere Eigenschaften)

1. *Die Splinekurve $S_m\mathbf{d}$ ist eine Abbildung $[t_{m+1}, t_{n+1}] \to \mathbb{R}^d$, "lebt" also sozusagen nur auf diesem Bereich. Daher müssen die* ***Randknoten*** $t_1, \ldots, t_{m+1}$ *und* $t_{n+1}, \ldots, t_{m+n+1}$ *eine besondere Rolle spielen.*

2. *Die erscheint zuerst einmal ein wenig seltsam, denn die Definition der B–Splines und damit auch der Splinekurve als Linearkombination von B–Splines mit Vektor–Koeffizienten gilt ja eigentlich auf ganz $\mathbb{R}$. Der Algorithmus 3.7 hingegen scheint*[30] *nur auf dem Intervall $[t_{m+1}, t_{n+1}]$ zu funktionieren.*

3. *In vielen, wenn nicht so gar den meisten Anwendungen und in der allermeisten Theorie wählt man die Randknoten als $m + 1$–fache Knoten, das heißt,*

$$t_1 = \cdots = t_{m+1}, \qquad t_{n+1} = \cdots = t_{n+m+1}.$$

   Das garantiert, daß am Rand des "guten" Intervalls die Splines gerade den ersten bzw. letzten Kontrollpunkt passieren, weswegen man diese Eigenschaft auch als ***Endpunktinterpolation***[31] *bezeichnet. Aber es ist sogar noch besser: Die Ableitungen am Rand*[32] *sind Vielfache der entsprechenden Differenzen der Rand–Kontrollpunkte. Somit enthält das Kontrollpolygon also wirklich jede Menge geometrische Information über die Kurve.*

4. *Es ist normalerweise nicht so einfach, sich das ganze Indexgewurstel des de–Boor–Algorithmus einzuprägen und im Gedächtnis zu behalten. Was man sich aber leicht merken kann, ist die Geometrie des Verfahrens: Das Verhältnis, in dem x gewisse Knotenintervalle aus $m + 1$, m, $m - 1$, ... und schließlich 2 Knoten teilt, wird auf passende Seiten des Kontrollpolygons übertragen, was zu neuen Kontrollpunkten führt.*

5. *Der de–Boor–Algorithmus verwendet ausschließlich* ***Konvexkombinationen*** *von Kontrollpunkten, um neue Kontrollpunkte zu bestimmen und deswegen ist die Splinekurve immer automatisch in der konvexen Hülle des Kontrollpolygons enthalten.*

Jetzt versuchen wir aber noch, die scheinbar widersprüchliche Situation aufzulösen, daß die Splinekurve mittels B–Splines *global* definiert ist, der Algorithmus die Kurve aber nur lokal beschreibt. Dafür betrachten wir zuerst einmal

[30] Man beachte die Wortwahl.

[31] Auch als *endpoint interpolation* bekannt und geschätzt.

[32] Und das sind *Vektoren*, da ist die Richtung viel wichtiger als der Absolutbetrag – willkommen in der wunderbaren Welt der Kurven!

die "konstante" Kurve, bei der alle $\mathbf{d}_j$ Skalare sind und denselben Wert haben, sagen wir $\mathbf{d}_j = 1$. Hier bekommen wir

$$\mathbf{d}_j^k(x) = \alpha(x)\,\mathbf{d}_{j-1}^{k-1}(x) + (1-\alpha(x))\;\mathbf{d}_j^{k-1}(x) = \alpha(x) + 1 - \alpha(x) = 1,$$

also

$$\sum_{j=1}^{n} N_j^k(\cdot \mid T) = 1. \tag{3.6}$$

Allerdings gilt diese Identität nur auf dem Intervall $[t_{m+1}, t_{n+1}]$, siehe Abb. 3.3,

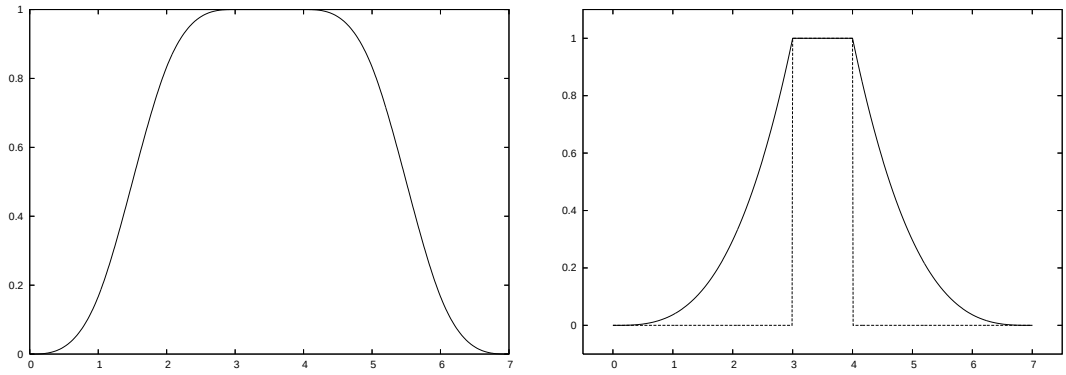


Abbildung 3.3: Plot der Funktion aus (3.6) für die Knotenfolge $\{0,1,2,3,4,5,6,7\}$ *(links)* – die Identität ist tatsächlich nur im Intervall $[3,4]$ erfüllt, das gerade von "innersten" Knoten bestimmt wird.

Mit dreifachen Randknoten gibt es einen nicht differenzierbaren Sprung an 3 and 4, während die Knotenfolge $\{3,3,3,3,4,4,4,4\}$ einen scharfen Sprung liefert *(rechts)*.

so daß unser Dilemma eine verblüffend einfache Lösung hat:

> Splinekurven sind überall wohldefiniert, aber wirklich "brav" sind sie nur auf dem Intervall $I_m(T) := [t_{m+1}, t_{n+1}]$.

Was uns schließlich ein weiteres gutes Argument für $m+1$–fache Randknoten liefert.

3.4 Ein wichtiger Spezialfall: De Casteljau und Bézier

Es gibt einen besonders einfachen Spezialfall von Knotenfolgen, nämlich $m+1$–fache Randknoten rechts und links und sonst nichts:

$$T = \{\underbrace{t_1, \ldots, t_1}_{m+1}, \underbrace{t_{n+1}, \ldots, t_{n+1}}_{m+1}\} =: \left\{t_1^{m+1}, t_{n+1}^{m+1}\right\}, \qquad n = m+1,$$

wobei wir dann gleich 0 und 1 als Randknoten wählen können, also[33] $T = \{0^{m+1}, 1^{m+1}\}$.

3.5 Curry, Schoenberg und andere Basen

Das fundamentalste Resultat der gesamten Splinetheorie besagt, daß B–Splines nicht irgendwelche stückweisen Polynome sind, sondern eine Basis eines ganz besonderen Vektorraums von Kurven bilden.

Definition 3.9 (Splineraum) *Der **Splineraum** der Ordnung m zur Knotenfolge $T = T_{m,n}$, in Zeichen $\mathbb{S}_m(T)$, besteht aus allen Funktionen f mit den folgenden beiden Eigenschaften:*

1. *f ist eine* auf T *stückweise polynomiale Funktion* vom (Höchst-) Grad[34] m *:* $f|_{(t_j, t_{j+1})} \in \Pi_m$.
2. *f besitzt eine gewisse globale **Glattheit***[35]

$$t_{j-1} < t_j = \cdots = t_{j+\mu-1} < t_{j+\mu} \qquad \Rightarrow \qquad f \in C^{m-\mu}(t_{j-1}, t_{j+\mu}). \tag{3.7}$$

Mit anderen Worten:

> Der Splineraum $\mathbb{S}_m(T)$ besteht aus allen Funktionen, deren Einschränkung auf **Knotenintervalle** Polynome vom Grad m sind und die an an einem Knoten der Vielfachheit μ immer noch $m - \mu$–mal stetig differenzierbar sind.

Bemerkung 3.10 *Sehen wir uns einmal die beiden Extremfälle an:*

[33] Der „Exponent" steht hier natürlich für die Vielfachheit der Knoten.

[34] Mit dem Grad von Polynomen ist das so eine Sache: Für manche ist es der Höchstgrad, für manche muß dann auch der Koeffizient zu diesem Grad von Null verschieden sein. Dswegen verwenden wir hier die formale Schreibweise mit $p \in \Pi_m$, was dann uneingeschränkt $\deg p \leq m$ bedeutet.

[35] Im Sinne von Differenzierbarkeit, zu anderen Glattheitsbegriffen kommen wir später noch.

$\mu = 1$: *an einem einfachen Knoten schließen die beiden Polynomstücke zu den angrenzenden Intervallen* $(m-1)$*–mal stetig differenzierbar aneinander an. Und das ist auch gut so, daß der Spline da nicht noch glatter ist, denn wenn zwei Polynome* p *und* q *vom Grad* m*, an einer Stelle* ξ *in allen Ableitungen bis Ordnung* m *übereinstimmen, dann brauchen wir nur eine Taylorentwicklung anzusehen und erhalten, daß*

$$p(x) = \sum_{j=0}^{m} \frac{p^{(j)}(\xi)}{j!}\,(x-\xi)^j = \sum_{j=0}^{m} \frac{q^{(j)}(\xi)}{j!}\,(x-\xi)^j = q(x)$$

ist und die beiden Polynome somit übereinstimmen müssten – was für einen Übergang beliebiger Glattheit an ξ *sorgen würde. Nein, nein,* $m-1$ *reicht völlig.*

$\mu = m+1$: *an einem Knoten* maximaler *Vielfachheit beträgt die Differenzierbarkeitsordnung* $m-(m+1) = -1$*, was man gerade mal so interpretieren kann, nämlich als einen Sprung – nullmalige stetige Differenzierbarkeit ist ja schließlich immer noch die gute alte Stetigkeit. Deswegen ist es auch nicht sinnvoll, höhere Vielfachheiten als* $m+1$ *zuzulassen, ganz abgesehen davon, daß die einem ohnehin nur die Rekursionsformel (3.3) versauen.*

Satz 3.11 (Curry & Schoenberg) *Die B–Splines* $N_j^m(\cdot\,|\,T)$ *sind eine* ***Basis*** *des Splineraums* $\mathbb{S}_m(T)$*.*

Eine Methode, Satz 3.11 zu beweisen, bestünde darin, die Ableitungsformel

$$\frac{d}{dx}N_j^m(x|T) = \frac{m}{t_{j+m}-t_j}N_j^{m-1}(x\,|\,T) - \frac{m}{t_{j+m+1}-t_{j+1}}N_{j+1}^{m-1}(x\,|\,T)\,, \qquad j = 1,\ldots,n, \tag{3.8}$$

zu verwenden, die man durch sehr elementare[36] aber etwas aufwendige Rechnung verifizieren kann, indem man folgendem Kochrezept z.B. aus (Sauer, 2000a) folgt: Man nehme die Ableitung der Rekursion (3.3), benutze (3.8) induktiv für die so erhaltenen Splines der Ordnung $m-1$ und fasse die Terme passend[37] zusammen. Nachdem die B–Splines stückweise Polynome sind, kann man sich bei (3.8) erst mal auf $x \in \mathbb{R} \setminus T$ beschränken, wo die Splines unendlich oft differenzierbar sind und dann mit einem Stetigkeitsargument auf ganz $\mathbb{R}$ fortsetzen. Das ist, wie gesagt, wie wir es nicht machen. Im späteren Theorieteil werden wir einen sehr eleganten Beweis kennenlernen, der auf dem "Blossoming Principle" beruht.

Aber die wesentliche Konsequenz von Satz 3.11 ist natürlich die folgende Beobachtung:

[36] Es gibt leider keine gute deutsche Übersetzung des englischen "straightforward".

[37] Na gut, viele Beweise bestehen darin, Terme richtig zusammenzufassen.

Jede Funktion $f \in \mathbb{S}_m(T)$ kann auf **eindeutige** Weise als

$$f = \sum_{j=1}^{n} f_j \, N_j^m(\cdot \,|\, T)$$

geschrieben werden. Insbesondere ist $\dim \mathbb{S}_m(T) = n$, hängt also im wesentlichen von der *Anzahl der Knoten*, nicht von der *Ordnung der Splines* ab.

Es gibt noch eine andere Basis für $\mathbb{S}_m$, nämlich die abgebrochenen Potenzen[38], die wir der Einfachheit halber nur für einfache Knoten betrachten wollen[39]. Die **abgebrochene Potenz** der Ordnung m mit Bruchtstelle t ist die Funktion

$$(x-t)_+^m = \begin{cases} (x-t)^m, & x \geq t, \\ 0, & x < t, \end{cases} \qquad x \in \mathbb{R},$$

und die Basis besteht nun aus den Monomen $1, x, \ldots, x^m$ und den abgebrochenen Potenzen $(\cdot - t_j)$, $j = m+1, \ldots, n-1$. Das sind dann wieder n stückweise polynomiale linear unabhängige Funktionen und damit eine Basis des Splineraums. Auch wenn diese Basis wieder nur für das Intervall $[t_{m+1}, t_n]$, uneingeschränkt vernünftig ist, haben die meisten der Basisfunktionen dennoch einen sehr großen Träger – eben ganz im Gegensatz zu den B–Splines.

Übung 3.2 Zeigen Sie:

1. Die abgebrochenen Potenzen $(\cdot - t)_+^m$ sind $m-1$–mal stetig differenzierbar.

2. Die Polynome und die abgebrochenen Potenzen sind linear unabhängig.

♢

3.6 Wie stellt man einen Spline dar?

Wir wollen diesen ersten Abschnitt damit beenden, uns zu überlegen, wie man nun so eine Splinekurve wirklich am Computer darstellt, insbesondere in `Octave`, denn ein Ziel dieses Kapitels soll die Erstellung einer kleinen "Mini–Spline–Toolbox" sein. Und die Kontrollpunkte $\mathbf{d}_j \in \mathbb{R}^d$, $j = 1, \ldots, n$, organisiert

[38] Die deutsche Übersetzung "Kastratfunktionen" die de Boor in seiner Vorlesung an der ETH Zürich (Boor, 1990) zuerst verwenden wollte, fand leider (?) zu wenig Zuspruch.

[39] Wir verwenden die Basis sowieso nicht, warum also unnötigen Aufwand betreiben.

man am besten in einer Matrix[40]

$$\mathbf{d} = [\mathbf{d}_1 \cdots \mathbf{d}_n] = \begin{bmatrix} d_{1,1} & \dots & d_{n,1} \\ \vdots & \ddots & \vdots \\ d_{1,d} & \dots & d_{n,d} \end{bmatrix} \in \mathbb{R}^{d\times n}, \qquad \mathbf{d}_j = \begin{bmatrix} d_{j,1} \\ \vdots \\ d_{j,d} \end{bmatrix}.$$

Um nun einen Spline an einer (endlichen) Punktmenge $X \subset \mathbb{R}$ auszuwerten, die wir geschickterweise in `Octave` durch einen Vektor repräsentieren werden[41], berechnen wir die Matrix[42]

$$\mathbb{R}^{n\times \#X} \ni \mathbf{N}_m(X) = \mathbf{N}_m(X \mid T) := \left[N_j^m(x \mid T) \;:\; \begin{matrix} j = 1,\dots,n \\ x \in X \end{matrix} \right],$$

und der Wert des Splines $S_m\mathbf{d}$ an X ist dann nur noch eine simple Matrixmultiplikation,

$$S_m\mathbf{d}(X) = \mathbf{d}\,\mathbf{N}_m(X). \tag{3.9}$$

Dieses einfache Stückchen Lineare Algebra wird sich im Folgenden noch als sehr nützlich erweisen. Die Funktion, die die Matrix $\mathbf{N}_m(X \mid T)$ für gegebene m, T, X bestimmt, trägt den im Moment noch etwas seltsamen Namen `BSplVander`, dessen Bedeutung uns aber bald klar werden wird.

Schauen wir uns schließlich noch an, wie man eine Splinekurve wie in Abb 3.4 eingibt und plottet: Zuerst definieren wir die Knotenfolge (mit vierfachen Randknoten) und die **Kontrollmatrix**[43] als

```
octave> T = [ 0,0,0,0,1,2,3,4,4,4,4 ];
octave> d = [ 0 1 1 .5 0 0 1 ; 0 .4 .8 1 .8 .4 0 ];
```

[40] Die Notation, Kleinbuchstaben, wenn auch fette, für Matrizen zu verwenden, ist nicht gerade Standard, aber ein wenig muss man schon zwischen dem Konzept "Kontrollpolygon", also Vektor von Punkten im $\mathbb{R}^d$ und der Darstellung "Matrix" unterscheiden, auch wenn sich, wie wir bald sehen werden, der Matrizenkalkül sehr gut ausnutzen lässt.

[41] Was uns sogar die Verwendung eines **Multiset** ermöglicht, also einer Menge mit Vielfachheiten.

[42] Ein Wort zur Notation: "Normale" Matrizen sind immer als

$$A = \left[a_{jk} \;:\; \begin{matrix} j = 1,\dots,m \\ k = 1,\dots,n \end{matrix} \right] \in \mathbb{R}^{m\times n}$$

zu sehen, wobei also j den Zeilen- und k den Spaltenindex bezeichnet. Und in genau demselben Sinne bezeichnet bei einer allgemein indizierten Matrix

$$A = \left[a(x,y) \;:\; \begin{matrix} x \in X \\ y \in Y \end{matrix} \right] \in \mathbb{R}^{X\times Y} \simeq \mathbb{R}^{\#X\times \#Y}$$

x den Zeilen- und y den Spaltenindex. Fazit: Oben steht die Zeile, unten die Spalte!

[43] Also die von den Kontrollpunkten gebildete Matrix.

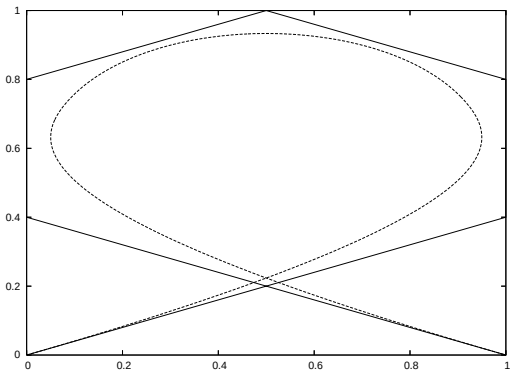


Abbildung 3.4: Eine Splinekurve und ihr Kontrollpolygon, alles ganz einfach – sogar die Knoten...

Um den Spline an einem feinen Gitter X auszuwerten, verwenden wir

```
octave> X = (0:.01:4); y = d * BSplVander( 3,T,X );
```

und Plotten bedeutet lediglich, `d` und `y` zeichnen zu lassen:

```
octave> hold on; plot( d(1,:),d(2,:) ); plot ( y(1,:), y(2,:) );
```

Und das war's auch schon!

3.7 Manipulation von Splines

In diesem Abschnitt befassen wir uns mit den elementaren Operationen, die man so auf Splinekurven anwenden kann.

3.7.1 Einfache geometrische Transformationen

Geometrische, zumindest affine, Transformationen einer Splinekurve sind sehr einfach, denn für $\mathbf{A} \in \mathbb{R}^{d\times d}$ und $\mathbf{y} \in \mathbb{R}^d$ erhalten wir, daß

$$\mathbf{A}S_m\mathbf{d} + \mathbf{y} = \sum_{j=1}^{n} (\mathbf{A}\mathbf{d}_j + \mathbf{y})\ N_j^m(\cdot\,|\,T) - \mathbf{y}\underbrace{\sum_{j=1}^{n} N_j^m(\cdot\,|\,T)}_{=1} + \mathbf{y},$$

so daß eine affine Transformation einer Kurve sich auf eine affine Transformation der Kontrollpunkte bzw. der Kontrollmatrix beschränkt. Und das ist nichts anderes als eine Linksmultiplikation[44] von $\mathbf{d}$.

3.7.2 Differentiation und Integration

In vielen Fällen ist es wichtig, die Ableitungen oder Stammfunktionen[45] einer Splinekurve zu bestimmen, gerade in Anwendungen mit "smoothing splines". Um eine Darstellung für die Ableitung einer Splinekurve herzuleiten, verwenden wir natürlich die Ableitunsformel (3.8) für die B–Splines und erhalten

$$\begin{aligned}
S_m\mathbf{d}'(x) &= \sum_{j=1}^{n} \mathbf{d}_j \left[\frac{m}{t_{j+m} - t_j} N_j^{m-1}(x \mid T) - \frac{m}{t_{j+m+1} - t_{j+1}} N_{j+1}^{m-1}(x \mid T) \right] \\
&= m \sum_{j=1}^{n} \frac{\mathbf{d}_j}{t_{j+m} - t_j} N_j^{m-1}(x \mid T) - m \sum_{j=2}^{n+1} \frac{\mathbf{d}_{j-1}}{t_{j+m} - t_j} N_j^{m-1}(x \mid T) \\
&= m \sum_{j=1}^{n+1} \frac{\mathbf{d}_j - \mathbf{d}_{j-1}}{t_{j+m} - t_j} N_j^{m-1}(x \mid T) \\
&=: S_{m-1}\mathbf{d}^1(x),
\end{aligned}$$

unter der Konvention $\mathbf{d}_0 = \mathbf{d}_{n+1} = 0$. Also bestimmt sich die Kontrollmatrix $\mathbf{d}^1$ der "Ableitungskurve" $S_{m-1}\mathbf{d}^1$ als

$$\mathbf{d}^1 = m\,\mathbf{d}\,\mathbf{D}_n \begin{bmatrix} t_{m+1} - t_1 & & \\ & \ddots & \\ & & t_{m+n+1} - t_{n+1} \end{bmatrix}^{-1} := m\,\mathbf{d}\,\mathbf{D}_n\,\Delta_m T \qquad (3.10)$$

wobei die **Differenzenmatrix D** als

$$\mathbf{D}_n = \begin{bmatrix} 1 & -1 & & \\ & \ddots & \ddots & \\ & & 1 & -1 \end{bmatrix} \in \mathbb{R}^{n \times n+1}$$

definiert ist und $\Delta_m T$ die Diagonalmatrix bezeichnet, die von den Reziprokwerten der m–Differenzen zwischen den Knoten gebildet wird:

$$\Delta_m T = \operatorname{diag}\left[(t_{j+m} - t_j)^{-1} \;:\; j = 1, \ldots, n+1 \right] \in \mathbb{R}^{n+1 \times n+1}.$$

[44] Für die, die die lineare Algebra schon wieder weitgehend vergessen haben: Es ist ein ziemlicher Unterschied, in welcher Reihenfolge bzw. von wo man Matrizen multipliziert.

[45] Auf Englisch "antiderivative".

Durch Iteration von (3.10) können wir nun auch Ableitungen höherer Ordnung berechnen, und zwar als

$$\begin{aligned} \mathbf{d}^2 &= \mathbf{d}\, m(m-1)\mathbf{D}_n\Delta_m T\, \mathbf{D}_{n+1}\Delta_{m-1}T, \\ &\vdots \\ \mathbf{d}^k &= \mathbf{d}\, \frac{m!}{(m-k)!} \prod_{j=0}^{k-1} \mathbf{D}_{n+j}\Delta_{m-j}T =: \mathbf{d}\,\mathbf{G}_k. \end{aligned}$$

Da für jedes $x \in \mathbb{R}$ und jede Kontrollmatrix $\mathbf{d}$ die Beziehung

$$\mathbf{d}\,\mathbf{N}_m^{(k)}(x) = \mathbf{d}^k\,\mathbf{N}_{m-k}(x \mid T) = \mathbf{d}\,\mathbf{G}_k\,\mathbf{N}_{m-k}(x \mid T)$$

gilt, erhalten wir, daß die Splinevektoren, also die Vektoren von Funktionen, die dadurch gebildet werden, daß man alle B–Splines "übereinanderstapelt", die Beziehung

$$\mathbf{N}_m^{(k)} = \mathbf{G}_k\,\mathbf{N}_{m-k}(\cdot \mid T), \tag{3.11}$$

erfüllen, die es uns erlaubt, diese Spaltenvektoren direkt miteinander zu verknüpfen.

Allerdings hätten[46] wir an dieser Stelle ein kleines Problem übersehen! Wenn wir Randknoten der Vielfachheit $m+1$ haben[47], dann ist $t_{m+1} = t_1$ and $t_{n+m+1} = t_{n+1}$, so daß $\Delta_m T$ dann die Gestalt

$$\Delta_m T = \begin{bmatrix} 0 & & & & \\ & * & & & \\ & & \ddots & & \\ & & & * & \\ & & & & 0 \end{bmatrix}^{-1}$$

hätte, was natürlich Unsinn ist, da wir versuchen würden, eine singuläre Matrix zu invertieren. Nichtsdestotrotz ist das nicht wirklich ein Problem, wenn wir erkennen, daß der erste B–Spline N_1^{m-1} ja nur auf (t_1, t_{m+1}) "lebt", also von Null verschieden ist, und so nichts zum Verhalten der Splinekurve und deren Ableitung, auf dem *relevanten Intervall* $I_m(T) = [t_{m+1}, t_{n+1}]$ beitragen kann. Dasselbe gilt natürlich auch für den Randknoten ganz rechts und den zugehörigen B–Spline N_{n+1}^{m-1}.

Folglich können wir, solange wir "nur" am Verhalten auf $I_m(T)$ interessiert sind[48], den ersten und den letzten B–Spline einfach ignorieren und erhalten so

[46] Oder haben.

[47] Und das war ja sozusagen unsere "Voreinstellung".

[48] Und das ist eine automatische Konsequenz aus $m+1$–facher Vielfachheit der Randknoten.

die etwas kompaktere Darstellung

$$(S_m \mathbf{d})' = S_{m-1} \widehat{\mathbf{d}}^1 \left(\cdot \,|\, \widehat{T} \right)$$

wobei $\widehat{T} = \widehat{T}_{m-1,n-1} = \{t_2, \ldots, t_{n+m}\}$ und

$$\widehat{\mathbf{d}}^1 = m \, \mathbf{d} \, \widehat{\mathbf{D}}_n \Delta_m \widehat{T}, \qquad \widehat{\mathbf{D}}_n = \begin{bmatrix} -1 & & \\ 1 & \ddots & \\ & \ddots & -1 \\ & & 1 \end{bmatrix} \in \mathbb{R}^{n \times n-1}.$$

Jetzt können wir die zugehörigen Matrizen $\widehat{\mathbf{G}}_k \in \mathbb{R}^{n \times n-k}$, $k = 1, \ldots, m$, rekursiv als

$$\widehat{\mathbf{G}}_0 = \mathbf{I}, \qquad \widehat{\mathbf{G}}_k = \widehat{\mathbf{G}}_{k-1} \, \widehat{\mathbf{D}}_{n-k+1} \Delta_{m-k+1} \widehat{T}_k,$$

mit Knotenfolgen

$$\widehat{T}_k = \widehat{T}_{k,n+m-k} = \{t_{k+1}, \ldots, t_{m+n+1-k}\},$$

berechnen und mittels

$$\widehat{\mathbf{d}}^k \mathbf{N}_{m-k} \left(\cdot \,|\, \widehat{T}_k \right) = \mathbf{d} \, \mathbf{N}_m^{(k)} \left(\cdot \,|\, T \right),$$

erhalten wir das folgende Gegenstück zu (3.11):

$$\mathbf{N}_m^{(k)} \left(\cdot \,|\, T \right) = \widehat{\mathbf{G}}_k \, \mathbf{N}_{m-k} \left(\cdot \,|\, \widehat{T}_k \right). \tag{3.12}$$

Fassen wir zusammen:

> Je nachdem, welches "Ableitungskonzept" wir betrachten wollen, gibt es zwei Möglichkeiten, die Ableitung einer Splinekurve zu bestimmen, eine, bei der sukzessive die äußersten Randknoten entfernt werden und eine, die diese erhält. Allerdings:
>
> 1. Auf dem "wesentlichen" Interval $I_m(T)$ stimmen beide Ansätze überein.
> 2. Für Randknoten der Vielfachheit $m+1$ müssen wir das zweite Konzept verwenden.

Wir können die Ableitungsformel auch verwenden um das (unbestimmte) Integral beziehungsweise die Stammfunktion einer Splinekurve zu berechnen, und zwar als

$$\int_{-\infty}^{x} S_m \mathbf{d}(t) \, dt = S_{m+1} \mathbf{d}^*(x), \qquad x \in [t_m, t_n], \tag{3.13}$$

wobei

$$\mathbf{d}_j^* = \sum_{k=1}^{j} \frac{t_{k+m+1} - t_k}{m+1} \mathbf{d}_k, \qquad j = 1, \ldots, n-1. \tag{3.14}$$

Diese Identität lässt sich recht einfach verifizieren, indem man die Ableitungsformel auf (3.13) anwendet – das ist eine nette kleine Übung.

Übung 3.3 Tun Sie's! ◊

3.7.3 Numerische integration von Splinefunktionen

Auch wenn uns die Formel (3.13) erlaubt, einen geschlossenen Ausdruck[49] für die Stammfunktion einer Splinekurve zu bestimmen, ist sie dennoch nur von beschränktem Nutzen für die Bestimmung von Integralen, die wir sehr oft in der Form

$$\int_{\mathbb{R}} N_j^m(x \mid T)\ N_k^m(x \mid T)\ dx$$

brauchen werden, die uns ganz natürlich im Kontext der "Least Squares Approximation" über den Weg laufen wird. Um Integrale numerisch zu berechnen, machen wir Gebrauch von zusammengesetzter **Gaußquadratur**, siehe (Gautschi, 1997; Isaacson & Keller, 1966; Sauer, 2000b). Zuerst einmal bemerken wir, daß das Produkt zweier Splinefunktionen[50] vom Grad m und m' wieder eine Splinefunktion, also eine auf T stückweise polynomiale Funktion, ist, allerdings mit dem lokalen Polynomgrad $m + m'$. Das heißt aber, daß es völlig ausreicht, auf jedem Knotenintervall $[t_j, t_{j+1}]$ eine Gauß–Quadraturformel der Ordnung m zu verwenden, denn so eine Quadraturformel integriert alle Polynome bis zum Grad $2m + 1$ *exakt*, also insbesondere auch das Produkt unserer Splines ohne irgendwelche Fehler jenseits der ebenso allgegenwärtigen wie unvermeidlichen Rundungsfehler zu machen.

Gut, seien als $\xi_0, \ldots, \xi_m \in [-1, 1]$ die **Gaußknoten** der Ordnung m für das Integral $\int_{-1}^{1} dx$, also die Nullstellen des Legendrepolynoms vom Grad $m + 1$, und seien $w_0, \ldots, w_m$ die zugehörigen Gewichte; wer diese Werte nicht selbst berechnen will, findet sie beispielsweise in (Abramowitz & Stegun, 1972). Da

$$\int_{t_j}^{t_{j+1}} f(x)\, dx = \frac{t_{j+1} - t_j}{2} \int_{-1}^{1} f\left(t_j + (x+1)\frac{t_{j+1} - t_j}{2}\right) dx,$$

[49] Und Ingenieure lieben geschlossene Ausdrücke!

[50] Das sind Splinekurve in $d = 1$. Wie soll man schließlich das Produkt zweier Kurven definieren? Als inneres oder als komponentenweises Produkt? Ist eigentlich beides nicht so widersinnig.

erhalten wir die gesuchte Quadratur als

$$\int_{t_j}^{t_{j+1}} f(x)\,dx \simeq \frac{t_{j+1}-t_j}{2} \sum_{k=0}^{m} w_k\, f\left(\frac{t_{j+1}+t_j}{2} + \frac{t_{j+1}-t_j}{2}\,\xi_k\right) \tag{3.15}$$

und das Integral über die gesamte Kurve kann dann ganz einfach durch Summation von $j = m+1, \ldots, n$ bzw. $j = 1, \ldots, n+m$ erhalten werden.

Auch dieser Prozess lässt sich wieder sehr effizient mittels Linearer Algebra implementieren[51], indem man den **Auswertungsvektor**

$$\Xi_j := \left[t_j^* + \frac{t_{j+1}-t_j}{2}\,\xi_k \;:\; k = 0, \ldots, m\right], \qquad t_j^* = \frac{t_{j+1}+t_j}{2}, \qquad j = m, \ldots, n,$$

definiert, sowie die Matrix

$$\mathbf{F} = \operatorname{diag}\left[\frac{t_{j+1}-t_j}{2} \;:\; j = m, \ldots, n\right] \left[f\left(\Xi_j\right) \;:\; j = m, \ldots, n\right], \tag{3.16}$$

und dann das Integral als

$$\int_{I_m(T)} f(x)\,dx \;=\; \sum_{j=m}^{n} \int_{t_j}^{t_{j+1}} f(x)\,dx = \mathbf{1}^T \mathbf{F} \mathbf{w}$$

berechnet, wobei $\mathbf{w} = [w_j \;:\; j = 0, \ldots, m]$ der Vektor der Gaußgewichte ist – und die erlauben es uns dann auch, hier das Gleichheitszeichen zu verwenden.

Natürlich ist (3.16) erst einmal nur für einfache Knoten korrekt, aber da die Intervalle zwischen zwei identischen Knoten zu einem Punkt degenerieren, gibt es da nicht viel zu integrieren und deswegen können sie einfach ignoriert werden.

[51] Zur Erinnerung: In `Octave` und natürlich auch in `Matlab` werden die Benutzer dazu angehalten, so viel Lineare Algebra wie möglich zu verwenden und beispielsweise Schleifen durch Vektorisierung zu ersetzen. Wenn man ehrlich ist, könnte man natürlich auch sagen, daß Schleifen so langsam implementiert sind, daß man eher von ihrem Gebrauch abgehalten wird.

Perilous to us all are the devices of an art deeper than we possess ourselves.

J. R. R. Tokien, *The Lord of the Rings – The Two Towers*

4 Interpolation

Die ursprüngliche „Definition“ eines Splines, aus der auch die Namensgebung resultiert, ist die eines interpolatorischen Kurvenlineals, zu Deutsch **Straklatte**. Was also liegt näher als sich erst einmal mit Interpolation zu befassen.

4.1 Interpolation und Minimalität

Es ist beinahe ein Glaubensgrundsatz der angewandten Mathematik, daß Splines für Interpolationszwecke geschaffen sind – schließlich ist der Namensgeber ja ein “interpolierendes” Kurvenlineal – und daß man deswegen mit ihnen immer gut interpolieren kann und sollte. Es gibt Argumente zugunsten dieser Religion, aber genauso gut Argumente, die diese Aussage nicht unterstützen. Aber fangen wir mal mit der “positiven” Seite an.

4.1.1 Interpolation an den Knoten und der natürliche Spline

Das **Interpolationsproblem** besteht darin, zu vorgegebenen **Interpolationsstellen** $X \subset \mathbb{R}$ und Werten[52] $y \in \mathbb{R}^X$ eine Funktion f zu finden, so daß

$$f(X) = y, \qquad \text{d.h.} \qquad f(x) = y_x, \qquad x \in X, \tag{4.1}$$

erfüllt ist. Interpolation mittels eines linearen Funktionenraums wie $\mathbb{S}_m(T)$ kann man immer in ein Problem aus der linearen Algebra transformieren: Wenn wir nämlich den Spline bzw. die Splinekurve in der B–Spline–Darstellung schreiben,

[52] Die Notation $\mathbb{R}^X$ mag ein wenig exzentrisch erscheinen, aber erstens ist es besser, die Dinge “natürlich” zu indizieren anstatt die Punkte durchzunumerieren und dann eine Zahl N zu haben, die nicht wirklich etwas aussagt, und zweitens ist $A^B := \{f : A \to B\}$ durchaus keine ungebräuchliche Schreibweise.

dann nimmt das Interpolationsproblem die folgende Gestalt an,

$$\sum_{j=1}^{n} \mathbf{d}_j \, N_j^m(x \mid T) = \mathbf{y}_x \qquad x \in X$$

die wir in Matrixform als

$$\mathbf{d}\,\mathbf{N}_m(X) = \mathbf{y}$$

schreiben können. Und das ist sogar die Form für *Kurven* und *Vektordaten* $\mathbf{y} \in \mathbb{R}^{d \times \#X}$, aber natürlich muss man wieder aufpassen, was man von welcher Seite multipliziert.

Die Matrix $\mathbf{N}_m(X)$ oder ihre Transponierte[53] heißt **Vandermondematrix**[54] des Interpolationsproblems; das erklärt auch den Namen der Funktion `BSplVander`, die wir ja vorher schon kennengelernt haben und die gerade diese Matrix erstellt hat, in der Basisfunktionen an Punkten ausgewertet wurden. Eine recht triviale aber bedeutsame und immer wieder auf wundersame Art und Weise wiederentdeckte Tatsache ist nun:

> Das Interpolationsproblem hat genau dann eine eindeutige Lösung wenn $\mathbf{N}_m(X)$ invertierbar ist.

Wenn wir mit Splines interpolieren wollen, dann drängen sich als Interpolationsstellen natürlich die einzigen ausgezeichneten Parameterwerte auf, die wir haben – die Knoten T. Damit ist es sicherlich keine schlechte Idee, an den **relevanten Knoten** $t_{m+1}, \ldots, t_{n+1}$ zu interpolieren, aber leider auch keine so uneingeschränkt gute, denn es sind ja nur insgesamt $n - m + 1$ Punkte, so daß die Matrix $\mathbf{N}_m(\{t_{m+1}, \ldots, t_{n+1}\})$ noch nicht einmal quadratisch, geschweige denn invertierbar ist. Allerdings: Es könnte schlimmer kommen, denn immerhin handelt es sich ja "nur" um ein *unterbestimmtes* Gleichungssystem

$$S_m\mathbf{d}\,(t_{m+j}) = \mathbf{y}_j, \qquad j = 1, \ldots, n - m + 1,$$

wir können also auf die Existenz *mindestens einer* Lösung hoffen[55], die sich dann vielleicht durch zusätzliche Bedingungen eindeutig machen lässt.

> Muss noch ausdrücklich gesagt werden, daß all dies nur für **einfache** Knoten gilt?

[53] Je nachdem, welcher Schule man angehört, aber über eine Transposition zu streiten ist doch eher Musikern vorbehalten.

[54] Das Interessante an der Geschichte ist, daß Vandermonde gar nichts damit zu tun hatte, sondern daß der Erfinder des Namens, wahrscheinlich Lagrange, nur unbedingt einen französischen Mathematiker ehren wolte – oh wunderbare Welt der Namensgebung.

[55] Na gut, hoffen kann man immer.

Also fügen wir unserem Interpolationsproblem weitere Bedingungen hinzu, vorzugsweise an den Endpunkten. Ist nun m eine ungerade Zahl, dann ist $m-1$ gerade und die Zusatzbedingungen lassen sich gleichmäßig und *symmetrisch* auf die Endpunkte verteilen[56]. Typische Beispiele für solche Randbedingungen sind:

Natürliche Randbedingungen: Ableitungen von *hoher Ordnung* sollen am Rand verschwinden, also

$$(S_m \mathbf{d})^{(k)}(t_j) = 0, \qquad k = \frac{m-1}{2} + 1, \ldots, m-1, \quad j = m+1, n+1.$$

Was daran nun so "natürlich" ist, erschließt sich (natürlich) auf den ersten Blick nicht so ganz unmittelbar!

Hermite Randbedingungen: Man legt Werte für die *Ableitungen* am Rand fest:

$$S_m^{(k)} \mathbf{d}(t_j), \qquad k = 1, \ldots, \frac{m-1}{2}, \quad j = m+1, n+1.$$

Das kann durchaus praktisch relevant sein, indem man, beispielsweise bei einer Werkzeugmaschine, Geschwindigkeit und Beschleunigung festlegt.

Periodische Randbedingungen: Man "schließt" die Kurve periodisch:

$$S_m^{(k)} \mathbf{d}(t_{m+1}) = S_m^{(k)} \mathbf{d}(t_{n+1}), \qquad k = 1, \ldots, m-1.$$

Wo es angemessen ist, ist das durchaus keine schlechte Idee.

Not–a–knot–Bedingungen: Diese "exzentrischsten" Randbedingungen, die eigentlich nur im Kontext kubischer ($m = 3$) Splines zu finden sind, fordert, daß die Splinekurve an den äußersten inneren Knoten t_{m+1} und t_{n+1} nicht nur $m-1$–mal, sondern m–mal und somit unendlich oft differenzierbar ist, daß also diese Knoten eben keine Knoten mehr sind.

Abgesehen von den natürlichen Randbedingungen fordern alle anderen Zusatzbedingungen zusätzliche Annahmen: Im Hermitefall müssen Ableitungen an den Endpunkten vorgegeben und nötigenfalls geraten werden, und die periodischen Randbedingungen sind ohnehin nur dann sinnvoll, wenn die zu interpolierenden Werte ebenfalls periodisch sind, d.h., wenn $\mathbf{y}_1 = \mathbf{y}_{n-m+1}$ ist.

[56] Das ist der Grund für die manchmal immer noch zu findende Legende, es gäbe nur Splines von ungeradem Polynomgrad.

4.1.2 Minimalität des natürlichen Splines

Es gibt noch eine weitere Methode, unterbestimmte Systeme zu lösen, indem man die Eindeutigkeit der Lösung durch *Minimierung* erzwingt: Man sucht sich unter allen Lösungen des Gleichungssystems diejenige heraus, die für ein bestimmtes Funktional[57] einen minimalen Wert ergibt. Anstatt also einfach nur nach einer Lösung von (4.1) zu suchen, *minimiert* man ein passend gewähltes Funktional *unter der Nebenbedingung* (4.1). Wir werden uns diesem Ansatz später noch in größerer Allgemeinheit widmen. Hier werden wir lediglich festhalten[58], daß der **natürliche Spline** in der Tat die Lösung eines Minimierungsproblems ist.

Zu diesem Zweck definieren wir die **Energie–Halbnormen**

$$|f|_k := \left(\int_{I_m(T)} \left(f^{(k)}(x) \right)^2 \right)^{1/2} dx, \qquad k \in \mathbb{N},$$

und legen fest, daß ein **natürlicher Splineinterpolant** derjenige *natürliche* Spline ist, der die Interpolationsbedingung

$$S_{m,T} f(t_j) = f(t_j), \qquad j = m+1, \ldots, n+1$$

an den Knoten erfüllt.

Satz 4.1 *Sei* $m = 2r+1$ *eine ungerade Zahl und* $f \in C^{r+1}(\mathbb{R})$. *Dann ist*

$$|S_{m,T} f|_{r+1} \le |f|_{r+1}. \tag{4.2}$$

In etwas prosaischeren Worten können wir Satz 4.1 folgendermaßen formulieren:

> Unter **allen** $C^{(m+1)/2}$–Lösungen des Interpolationsproblems an den Stellen $t_{m+1}, \ldots, t_{n+1}$ minimiert der natürliche Spline das Energiefunktional $|\cdot|_{r+1}$.

Das erklärt dann übrigens auch den Namen "natürlicher Spline", denn ein Spline ist eigentlich ein flexibles Kurvenlineal, das durch Gewichte gezwungen wird, durch gewisse vorgegebene Punkte zu verlaufen und das dann die energetisch günstigste Position einnimmt, indem es das Energiefunktional

$$\int_I (\kappa(f))^2 \, ds \simeq \int_I |f''(t)|^2 \, dt, \qquad \kappa(f) = \frac{d^2}{ds^2} f,$$

[57] Oftmals als "Energiefunktional" bezeichnet, auch wenn es manchmal gar nicht wirklich etwas mit Energie in einem physikalischen Sinn zu tun hat. Aber es gibt einem das gute Gefühl, etwas besonders ökonomisches zu tun.

[58] Und diese Tatsache und deren Beweis dürfen in einer Vorlesung über Splines einfach nicht fehlen.

Abbildung 4.1: Ein "echter" Spline. Die Gewichte (und sie tragen ihren Namen zu Recht) fixieren das felxible Lineal an den Interpolationspunkten. Danke an Dr. M. Hollenhorst, der der Arbeitsgruppe Numerik seinen Spline überlassen hat.

minimiert – wobei das "≃" wirklich für eine sehr, sehr grobe Näherung steht. Mit anderen Worten:

Der kubische natürliche Spline ist **kein** Spline!

Da der Beweis von Satz 4.1 elementar und recht erhellend ist, schauen wir uns ihn doch auch einmal an:

Beweis von Theorem 4.1: Unter Verwendung der Abkürzungen $Sf = S_{m,T}f$ und $I = I_m(T)$ beginnen wir mit

$$\begin{aligned} |f - Sf|_{r+1}^2 &= \int_I \left(f^{(r+1)}(x) - (Sf)^{(r+1)}(x)\right)^2 dx \\ &= \int_I \left(f^{(r+1)}(x)\right)^2 - 2f^{(r+1)}(x)(Sf)^{(r+1)}(x) + \left((Sf)^{(r+1)}(x)\right)^2 dx \\ &= |f|_{r+1,I}^2 - 2\int_I \left(f^{(r+1)}(x) - (Sf)^{(r+1)}(x)\right)(Sf)^{(r+1)}(x)\, dx - |Sf|_{r+1}^2 . \end{aligned} \tag{4.3}$$

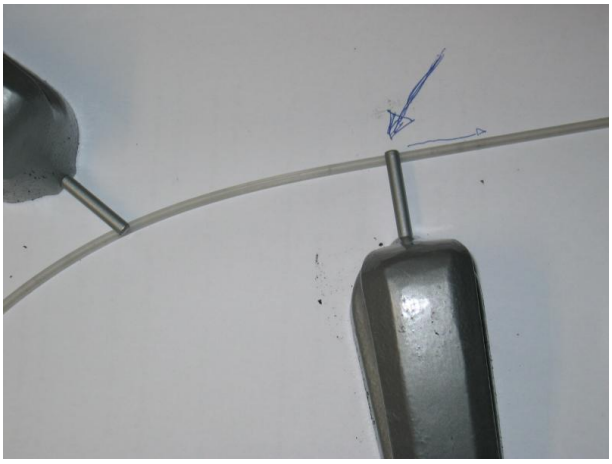


Abbildung 4.2: Die natürlichen Randbedingungen des kubischen Splines: Die freien Enden nehmen, sich selbst überlassen, eine lineare Gestalt an.

Für $j = m+1, \ldots, n$ liefert partielle Integration

$$
\begin{aligned}
&\int_{t_j}^{t_{j+1}} \left(f^{(r+1)}(x) - Sf^{(r+1)}(x)\right) Sf^{(r+1)}(x)\, dx \\
&= \left(f^{(r)}(x) - Sf^{(r)}(x)\right) Sf^{(r+1)}(x)\Big|_{t_j}^{t_{j+1}} - \int_{t_j}^{t_{j+1}} \left(f^{(r)}(x) - Sf^{(r)}(x)\right) Sf^{(r+2)}(x)\, dx \\
&= \sum_{l=0}^{k} (-1)^{r-l} \left(f^{(r-l)}(x) - Sf^{(r-l)}(x)\right) Sf^{(r+l+1)}(x)\Big|_{t_j}^{t_{j+1}} \\
&\qquad + (-1)^{k+1} \int_{t_j}^{t_{j+1}} \left(f^{(r-k)}(x) - S_f^{(r-k)}(x)\right) \underbrace{Sf^{(r+k+2)}(x)}_{=0 \text{ für } k=r}\, dx, \qquad k = 1, \ldots, r \\
&= \sum_{l=0}^{r} (-1)^{r-l} \left(f^{(r-l)}(x) - Sf^{(r-l)}(x)\right) Sf^{(r+l+1)}(x)\Big|_{t_j}^{t_{j+1}},
\end{aligned}
$$

und wenn wir das nun über j summieren, dann heben sich die gesamten inneren

Terme weg und es bleibt nur noch

$$\int_I \left(f^{(r+1)}(x) - Sf^{(r+1)}(x)\right) Sf^{(r+1)}(x)\, dx$$
$$= \sum_{l=0}^{r} (-1)^{r-l} \left(f^{(r-l)}(x) - Sf^{(r-l)}(x)\right) S_f^{(r+l+1)}(x) \Big|_{t_m}^{t_{n+1}} = 0$$

übrig. Wenn wir das nun wieder in (4.3) einsetzen, dann landen wir bei

$$|Sf|_{r+1}^2 = |f|_{r+1}^2 - |f - Sf|_{r+1}^2 \le |f|_{r+1}^2$$

mit Gleichheit dann und nur dann wenn $f - Sf \in \Pi_r$. □

4.1.3 Schoenberg, Whitney, Greville

Im vorherigen Abschnitt haben wir darauf bestanden, an den "relevanten" Knoten zu interpolieren und dafür den Preis weiterer, mehr oder weniger natürlicher oder künstlicher Zusatzbedingungen bezahlt, um für unser Interpolationsproblem eine *eindeutige* Lösung zu bekommen. Es ist aber unabhängig davon eine interessante Frage, wie die **Interpolationsstellen** $x_1, \ldots, x_n$ zu legen sind, damit wir mit dem n–dimensionalen Vektorraum $\mathbb{S}_m(T_{m,n})$ interpolieren können. So einfach wie bei den Polynomen, wo es reichte, daß die Interpolationsstellen alle verschieden waren, ist es halt nicht mehr.

Es ist intuitiv klar, daß bei beliebiger Wahl von n Interplationsstellen Probleme auftreten müssen: Auf jedem Knotenintervall $[t_j, t_{j+1}]$ ist der Spline ein Polynom vom Grad höchstens m und wenn in einem solchen Intervall mehr als $m+1$ Punke zu liegen kommen, dann wird das Interpolationsproblem im allgemeinen unlösbar sein, weil bereits das *lokale* Teilproblem auf dem Intervall $[t_j, t_{j+1}]$ unlösbar ist. Daher muss es eine Beziehung zwischen den Interpolationsstellen und den Knoten geben, die glücklicherweise so einfach ist, wie man sie sich nur wünschen kann.

Satz 4.2 (Schoenberg & Whitney) *Interpolation an den Stellen* $X = \{x_j : j = 1, \ldots, n\}$, $x_1 < \cdots < x_n$ *hat genau dann*[59] *eine eindeutige Lösung in* $\mathbb{S}_m(T)$ *wenn*

$$t_j < x_j < t_{j+m+1}, \qquad j = 1, \ldots, m. \tag{4.4}$$

[59] Um genau zu sein: Das gilt so nur, wenn kein Knoten die maximal erlaubte Vielfachheit $m+1$ hat.

Noch leichter kann man sich die **Schoenberg–Whitney–Bedingung** (4.4) merken, wenn man sich ihre geometrische Bedeutung vor Augen führt:

> Jede der Interpolationsstellen x_j, $j = 1, \ldots, n$, hat an einer Stelle zu liegen, an der der B–Spline mit demselben Index positiv ist.

Ein vollständiger Beweis von Satz 4.2 geht über das hinaus, was wir an dieser Stelle brauchen[60], aber es ist ziemlich leicht einzusehen, daß und warum diese Bedingung notwendig ist. Nehmen wir zum Beispiel an, daß es einen Index j gäbe, so daß $x_j \le t_j$ ist, also auch $x_j \le t_k$, $k \ge j$, denn die Knoten sind aufsteigend angeordnet. Nachdem der B–Spline N_k^m als Träger das Intervall $[t_k, t_{k+m+1}]$ hat, folgt, daß

$$N_k^m(x_j \,|\, T) = 0, \qquad k = j, \ldots, n.$$

Ein Blick auf die ersten j Spalten der Vandermondematrix $\mathbf{N}_m(X)$ zeigt, daß

$$\begin{bmatrix} N_1(x_1 \,|\, T) & \ldots & N_1(x_j \,|\, T) \\ \vdots & \ddots & \vdots \\ N_{j-1}(x_1 \,|\, T) & \ldots & N_{j-1}(x_j \,|\, T) \\ N_j(x_1 \,|\, T) & \ldots & N_j(x_j \,|\, T) \\ \vdots & \ddots & \vdots \\ N_n(x_1 \,|\, T) & \ldots & N_n(x_j \,|\, T) \end{bmatrix} = \begin{bmatrix} N_1(x_1 \,|\, T) & \ldots & N_1(x_j \,|\, T) \\ \vdots & \ddots & \vdots \\ N_{j-1}(x_1 \,|\, T) & \ldots & N_{j-1}(x_j \,|\, T) \\ 0 & \ldots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \ldots & 0 \end{bmatrix}$$

und diese Spalten sind linear abhängig, weil sie aus j Vektoren bestehen, die nur in den ersten $j-1$ Komponenten von Null verschieden sind, also als j Vektoren im $\mathbb{R}^{j-1}$ aufgefasst werden können. Also ist die Matrix singulär. Im Falle der Existenz von j mit der Eigenschaft $x_j \ge t_{j+m+1}$ führt eine analoge Betrachtung der ersten j Zeilen der Matrix zum selben Ergebnis.

Also sagt uns Satz 4.2 ganz genau, wo wir die Interpolationsstellen hinzulegen haben, um Eindeutigkeit der Interpolation zu erreichen. Was es uns allerdings nicht sagt, ist, wie wir diese Stellen *systematisch* aus den Knoten bestimmen sollen – alles, was sich in den geforderten Intervallen befindet ist ja in Ordnung. Dennoch gibt es eine "kanonische" Wahl, nämlich die sogenannten **Greville–Abszissen**:

$$x_j = \frac{1}{m} \sum_{k=1}^{m} t_{j+k}, \qquad j = 1, \ldots, n. \tag{4.5}$$

Die Definition der Greville–Abszissen ist erstaunlicherweise unabhängig von den beiden "äußersten" Knoten t_1 und t_{n+m+1}, aber das soll uns nicht weiter

[60] Soooo schwer ist er aber auch wieder nicht, siehe (Sauer, 2000a).

stören, zumal diese beiden Knoten ohnehin keine besonders große Rolle spielen. Was man aber leicht sieht, ist, daß die Greville–Abszissen wirklich "gute" Inerpolationspunkte sind, denn es gilt

$$t_j \le t_{j+1} \le \frac{1}{m}\,(t_{j+1} + \cdots + t_{j+m}) \le t_{j+m} \le t_{j+m+1}$$

mit Gleichheit in der linken Ungleichung genau dann wenn $t_j = \cdots = t_{j+m}$ und Gleichheit in der rechten Ungleichung genau dann, wenn $t_{j+1} = \cdots = t_{j+m+1}$ gilt, das heißt, die **Schoenberg–Whitney–Bedingung** kann durch die Greville–Abszissen nur verletzt werden, wenn es innere Knoten der Vielfachheit $m+1$ gibt. Aber das ist für Interpolation ohnehin eher unsinnig, weil dann der Interpolant unstetig werden kann.

An einem Knoten der Vielfachheit $m+1$ hat die Splinekurve eine Unstetigkeit und daher kann ein Interpolationswert an dieser Kurve nicht festgelegt werden: Soll man den rechtsseitigen oder doch eher den linksseitigen Grenzwert wählen?

Eine weitere Rechtfertigung für die Benutzung der Greville–Abszissen liefert der **Schoenberg–Operator**

$$\mathscr{S}_m f := \sum_{j=1}^{n} f(x_j)\; N_j^m(\cdot \mid T)\,, \tag{4.6}$$

der die Funktion f nicht interpoliert, sondern ihre Werte an den Greville–Abszissen als Kontrollpunkte verwendet. Wir werden uns diesen äußerst nützlichen Operator später noch genauer ansehen.

4.1.4 Parametrische Interpolation – so einfach ist es nicht!

Soweit zur schönen Theorie, aber in der Praxis sind die Abszissen selten gegeben, alles was man dort kennt, sind Punkte $\mathbf{y}_1, \ldots, \mathbf{y}_n \in \mathbb{R}^d$, durch die die Kurve gehen soll, die beispielsweise daher kommen, daß ein Werkstück gescannt oder von einem CAM–System abgetastet wurde. Die zugehörigen Abszissen sind nicht festgelegt und können, nein, müssen irgendwie bestimmt werden, natürlich in Abhängigkeit von den Datenwerten. Für diese Abszissenwahl gibt es eine ganze Menge von Strategien, die alle das Aussehen und die Qualität der resultierenden interpolierenden Kurve teilweise recht dramatisch beeinflussen können; einige dieser Strategien sind in (Farin, 1988) aufgelistet, basieren aber alle mehr oder weniger auf heuristischen Kriterien.

Intuitiv[61] würde man fordern, daß der Abstand zwischen den Interpolationsstellen x_j dem Abstand zwischen den Datenpunkten $\mathbf{y}_j$ entsprechen sollte, was die Wahl

$$\begin{aligned} x_1 &= 0 \\ x_2 &= \left\|\mathbf{y}_2 - \mathbf{y}_1\right\|_2 \\ &\vdots \\ x_k &= x_{k-1} + \left\|\mathbf{y}_k - \mathbf{y}_{k-1}\right\|_2 = \sum_{j=2}^{k} \left\|\mathbf{y}_j - \mathbf{y}_{j-1}\right\|, \qquad k = 2, \ldots, n, \end{aligned}$$

liefert. Nun könnte man versuchen, die Knoten T so anzupassen, daß die Menge X der Interpolationsstellen nicht nur die **Schoenberg–Whitney–Bedingung** erfüllt, sondern sogar die der Greville–Abszissen ist. Bestehen wir außerdem noch auf $m+1$–fache Randknoten, dann erhalten wir die Knoten durch sukzessives Einsetzen in (4.5):

$$\begin{aligned} t_1 = \cdots = t_{m+1} &= x_1 \\ t_{m+2} &= m\, x_2 - \sum_{k=1}^{m-1} t_{k+2} \\ &\vdots \\ t_{m+j} &= m\, x_j - \sum_{k=1}^{m-1} t_{k+j}, \qquad j = 2, \ldots, n \end{aligned} \tag{4.7}$$

Allerdings liefert uns diese Vorgehensweise als erste Überraschung[62] keine $m+1$–fache Vielfachheit des rechten Randknotens mehr! Außerdem sieht (4.7) auch bei dem folgenden numerischen Beispiel sehr schlecht aus: Wir tasten den Einheitskreis gleichmäßig ab,

```
octave> y = [ cos( 2*pi*(0:6)/7 ); sin(  2*pi*(0:6)/7 ) ]
y =

   1.00000   0.62349  -0.22252  -0.90097  -0.90097  -0.22252   0.62349
   0.00000   0.78183   0.97493   0.43388  -0.43388  -0.97493  -0.78183
```

und bestimmen dann die Knotenfolge

```
octave> T=GrevilleKnots( 3,y )
ans =
```

[61] So, damit sind wir auch in der wunderbaren Welt der Heuristik!

[62] Und als Beleg dafür, daß hier etwas faul sein könnte!

```
 Columns 1 through 8:

  0.00000  0.00000  0.00000  0.00000  2.60330  2.60330  2.60330  5.20660

 Columns 9 through 11:

  5.20660  5.20660  5.20660
```

mit einem *dreifachen* Knoten im Inneren! Die zugehörigen Greville–Abszissen sind schließlich

```
octave> X = Greville( 3,T )
X =

   0.00000   0.86777   1.73553   2.60330   3.47107   4.33884   5.20660
```

und das ist eigentlich in Ordnung - der Abstand zwischen den Interpolationsstellen „passt“.

Allerdings ist die Lösung dieses Interpolationsproblems sehr enttäuschend – obwohl die Funktion in Abb. 4.3 an den Ecken des Polygons interpoliert, hat sie wegen des dreifachen Knotens einen Knick, und das sieht gar nicht schön aus. Mit anderen Worten: Die naive Strategie ist es leider nicht. Es gibt sogar

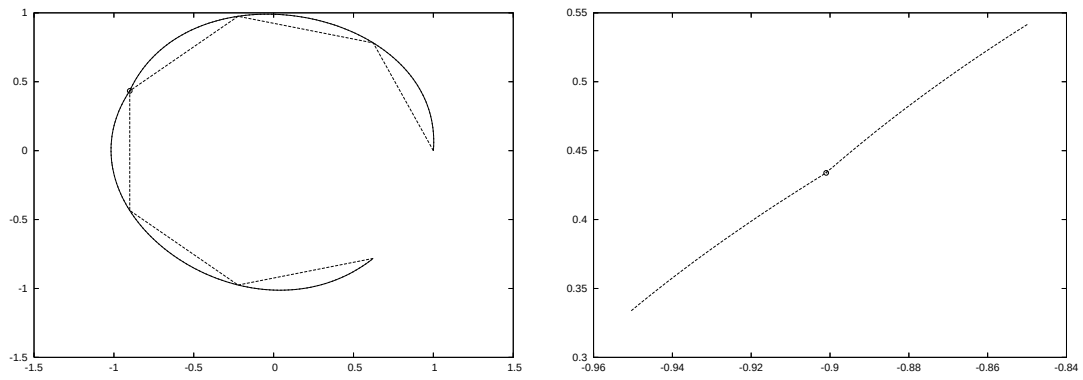


Abbildung 4.3: Interpolant, dessen Knoten nach (4.7) berechnet wurden. Der dreifache Knoten liefert eine Stelle, an der die Funktion nicht differenzierbar ist, siehe den Detailausschnit *(rechts)*.

Beispiele, bei denen eine *nicht aufsteigende* Knotenfolge generiert wird. Liegt das nun an der Einfachheit des Ansatzes oder ist das ein prinzipielles Problem? Schauen wir doch einfach nach!

Allgemein muss jeder zulässige **Knotenvektor**[63] $\mathbf{t} = [t_j : j = 2, \ldots, m+n]$ eine Lösung von

$$\frac{1}{m}\begin{bmatrix} 1 & \ldots & 1 & & \\ & \ddots & & \ddots & \\ & & 1 & \ldots & 1 \end{bmatrix}\mathbf{t} =: \mathbf{M}\mathbf{t} = \mathbf{x}, \qquad \begin{bmatrix} -1 & 1 & & \\ & \ddots & \ddots & \\ & & -1 & 1 \end{bmatrix}\mathbf{t} = \mathbf{D}^T\mathbf{t} \geq 0 \quad (4.8)$$

mit $\mathbf{M} \in \mathbb{R}^{n \times n+m-1}$ und $\mathbf{D}^T \in \mathbb{R}^{n+m-2 \times n+m-1}$ sein – das ist ein System von *linearen Ungleichungen*. die man in Tateinheit mit der Minimierung eines linearen Funktionals durch **Lineare Programmierung** lösen kann, siehe (Dantzig, 1963; Nocedal & Wright, 1999), aber auch (Karlin, 1959) für den hochinteressanten Bezug zur Spieltheorie. Beispielsweise könnten wir auch noch den Abstand zwischen den Knoten maximieren, was uns zum Minimierungsproblem

$$\max_{t,s} s, \qquad \begin{bmatrix} \mathbf{M} & \mathbf{0} \\ \mathbf{D}^T & -\mathbf{1} \end{bmatrix}\begin{bmatrix} \mathbf{t} \\ s \end{bmatrix}\begin{bmatrix} = \\ \geq \end{bmatrix}\begin{bmatrix} \mathbf{x} \\ \mathbf{0} \end{bmatrix}, \qquad \mathbf{t}, s \geq 0. \quad (4.9)$$

führt, das wir dann nur noch in einen Standardlöser[64] zu stecken brauchen. Das wird von der Funktion `GrevilleKnotsOpt` erledigt und die sagt uns auch, daß wir für diese Abtastung nichts besseres erwarten können:

```
octave> GrevilleKnotsOpt( 3,y )
*** Minimal Knot distance: 0.000000e+00 ***
ans =

 Columns 1 through 7:

   0.00000   0.00000   0.00000   2.60330   2.60330   2.60330   5.20660

 Columns 8 and 9:

   5.20660   5.20660
```

und das ist genau das, was uns unser naives Verfahren auch geliefert hat. Daß die Funktion ansonsten schon ordentlich funktioniert zeigt uns das Beispiel

```
octave> GrevilleKnotsOpt( 3,Greville( 3,[0 0 0 (0:5) 5 5 5 ] ) )
*** Minimal Knot distance: 0.000000e+00 ***
ans =
```

[63] Das ist nichts anderes als die Knotenfolge, nur eben jetzt als Vektor geschrieben und damit an Matrizen multipizierbar.

[64] Hier verwenden wir `lp_solve`, (Berkelaar *et al.*, 2000), ein wunderbar in `octave 2.9.x` zu integrierendes und auch sehr schnelles Tool zur linearen Optimierung. Alternativ könnte man auch auf `glpk`, (Makhorin, 2000), zurückgreifen, das automatisch in `Octave` integriert wird und in vielen Distributionen auch ist.

```
 Columns 1 through 7:

   0.00000   0.00000   0.00000   1.00000   2.00000   3.00000   4.00000

 Columns 8 through 10:

   5.00000   5.00000   5.00000
```

Es stellt sich sogar heraus, daß dieses Optimierungsproblem für viele Konfigurationen *keine* Lösung hat[65], daß es also keine Knotenfolge gibt, deren zugehörige Greville–Abszissen voneinander denselben Abstand haben wie die zu interpolierenden Punkte! Am einfachsten findet man solche Beispiele durch zufällige Wahl der Punkte:

```
octave> d = rand( 5,1 ); dd = [ 0, ( tril( ones(5) )* d )' ];
octave> GrevilleKnotsOpt( 3,dd )
warning: lp_solve: some elements in list of return values are undefined
Error: Problem unsolvable
ans = [](0x0)
```

Daher besteht der "Stadardansatz" darin, die *Knoten* so zu wählen, daß ihr Abstand dem Abstand der Datenpunkte entspricht und dann an den wohldefinierten Greville–Abszissen zu interpolieren. Deren Abstand ist dann zwar nicht mehr ganz so toll, aber zumindest funktioniert alles.

4.2 Was ist faul an Interpolation?

Also gut, Splines können interpolieren, ob nun an den Greville-Abszissen, an den Knoten oder anderswo, aber dennoch bleiben bei der Interpolation *unvermeidbare* Probleme übrig, die schlicht und einfach daher kommen, daß man im allgemeinen mit *glatten* Funktionen interpoliert.

4.2.1 Abhängigkeit von der Parametrisierung

Das erste Problem, das uns bei der Interpolation erwartet, ist die starke Abhängigkeit des Ergebnises von der *Parametrisierung* der Interpolationsstellen, also der Wahl, welchen Parameterwert wir welchem der zu interpolierenden Punkte zuordnen. Denn im Gegensatz zum funktionalen Fall ist die Zuordnung zwischen Interpolationsstelle und Interpolationswert nicht a priori festgelegt.

[65] Der sogenannte "zulässige Bereich", also die Lösungsmenge des Ungleichungssystems ist die leere Menge.

Wir werden dieses Phänomen mit einem sehr einfachen Beispiel illustrieren, nämlich wieder mit den sieben Punkten auf dem Einheitskreis von oben und kubischen Splines. Dazu brauchen wir $n + m + 1 = 7 + 3 + 1 = 11$ Knoten mit Randknoten der maximalen[66] Vielfachheit 4. Beginnen wir mit *gleichverteilten* Knoten, also

```
octave> T = [ 0 0 0 0 1 2 3 4 4 4 4 ];
octave> X = Greville( 3,T )
X =

  0.00000  0.33333  1.00000  2.00000  3.00000  3.66667  4.00000
```

wobei die Funktion `Greville` überraschenderweise die Greville-Abszissen zur Knotenfolge berechnet. Um zu interpolieren, stellen wir als nächstes die **Vandermondematrix** auf und berechnen die Kontrollpunkte mittels

```
octave> V = BSplVander( 3,T,X )'; d = ( V \ y' )';
```

Die Transposition in dieser Darstellung hat lediglich technische Gründe. Und in der Tat erhalten wir so das Ergebnis in Abb. 4.4. Gut, das ist nicht allzu schlecht,

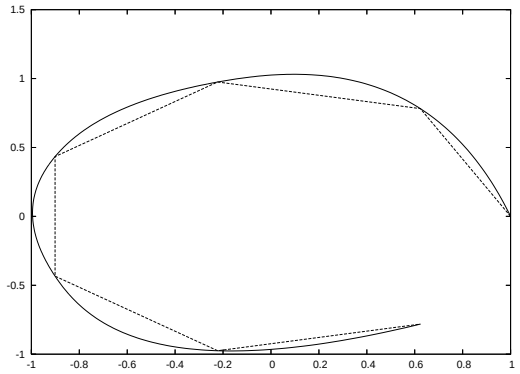


Abbildung 4.4: Der Splineinterpolant mit äquidistanten Knoten und den zugehörigen Greville-Abszissen. Man erkennt sehr schön, daß die Abweichung des Splines vom Datenpolygon dort größer wird, wo das Polygon deutlich kürzere Teilstücke hat. Anschaulich ist es auch klar, daß man an dieser Stelle für die unpassende Parametrisierung „bezahlt".

aber jetzt vergrößern wir den Abstand zwischen den Knoten und damit auch zwischen den Greville-Abszissen auf *nicht proportionale* Weise:

[66] Aus den altbekannten Gründen!

```
octave> T = [ 0 0 0 0 1 4 9 16 16 16 16 ];X = Greville( 3,T );
```

Es überrascht uns nun nicht mehr, daß der zugehörige Interpolant eine größere Abweichung im „späteren“ Teil der Kurve hat, während der Interpolant zu den immer dichteren Knoten

```
octave> T = [ 0 0 0 0 1.3 1.7 1.9 2 2 2 2 ];X = Greville( 3,T );
```

seine größte Abweichung im Anfangsstück der Kurve hat, siehe Abb. 4.5. Dies

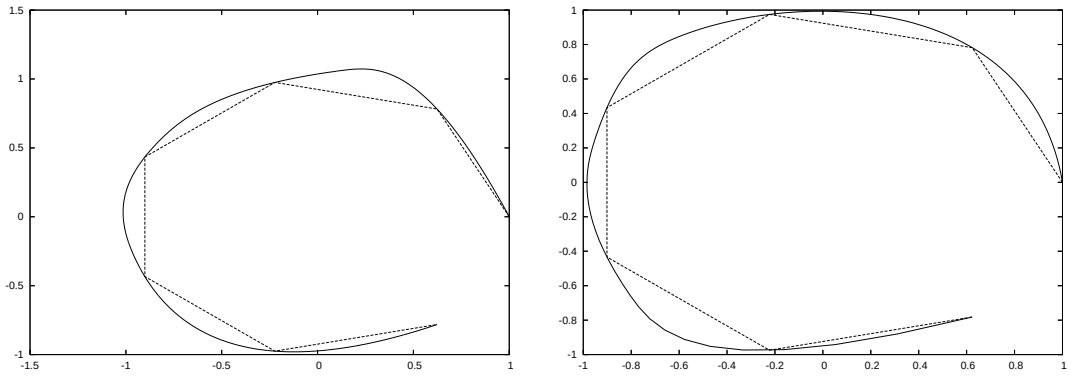


Abbildung 4.5: Die Interpolanten mit wachsendem *(links)* und fallendem *(rechts)* Abstand zwischen den Knoten.

sollte auch die letzten Zweifler davon überzeugen, daß das Ergebnis sehr sensibel für die Parametrisierung der Interpolationsstellen ist. Unglücklicherweise - und das hat uns das Desaster mit den Greville-Abszissen im letzten Kapitel leider recht deutlich aufgezeigt - ist es nicht einfach, gute[67] Parameterwerte für die Interpolationspunkte zu bestimmen. Fassen wir zusammen:

> Das Ergebnis der Splineinterpolation hängt ganz massiv von der Wahl der zugehörigen Parameterwerte ab.

4.2.2 Der Fluch der Interpolation - ein unerwünschter Überschuss

Die hässliche Seite der Interpolation durch glatte Funktionen zeigt sich, wenn man Ecken interpolieren will. Ein ganz einfaches Beispiel erhalten wir, wenn wir

[67] Und wir werden noch sehen, daß es schon gut wäre, Greville-Abszissen als Interpolationspunkte zu haben.

die Verbindungslinien $[0,0] \rightarrow [1,0]$ und $[1,0] \rightarrow [1,1]$ abtasten und diese Daten interpolieren. Legen wir, um einfach anzufangen, vier Interpolationspunkte auf jede der Linien,

```
octave> y = [ (0:1/3:1 ); zeros( 1,4 ) ];
octave> y = [ y, [ ones( 1,3 ); ( 1/3:1/3:1 ) ] ];
```

wählen die zu diesem Zweck notwendigen 11 gleichverteilten Knoten

```
octave> T = [ 0 0 0 (0:4) 4 4 4 ];
```

und berechnen den Interpolanten mit unserer „Standardmethode“

```
octave> d = ( BSplVander( 3,T,Greville( 3,T ) )'\y' )';
```

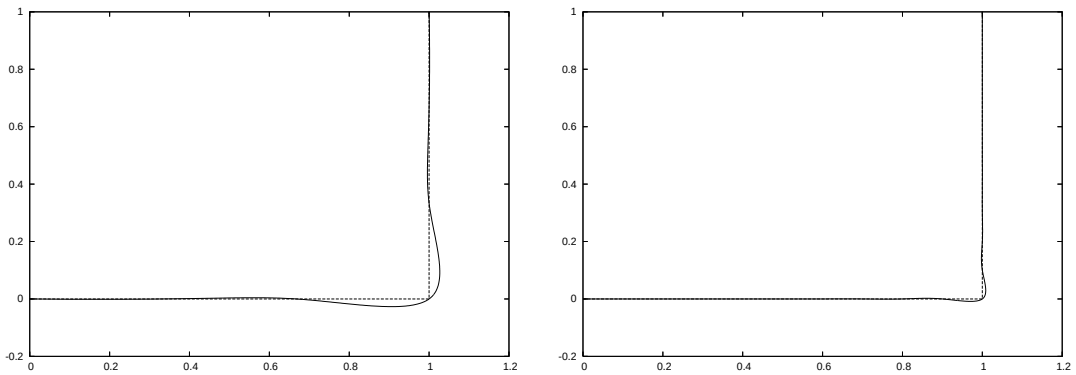


Abbildung 4.6: Das Überschießen des Splineinterpolanten in Ecken, und zwar mit vier *(links)* und zehn *(rechts)* Abtastpunkten auf der stückweise linearen Kurve.

Abb. 4.6 zeigt das Ergebnis dieses Interpolationsprozesses und das normalerweise unerwünschte Überschießen des Interpolanten. Natürlich wird der Effekt geringer, wenn man die Interpolationspunkte dichter wählt, aber er verschwindet nicht - und beispielsweise beim Fräsen oder Laserschneiden sind solche Effekte tödlich.

Was noch schlimmer als die Abweichung von der „Idealgestalt“ der Kurve an den Ecken ist, ist der Verlust von geometrischer Information[68]: Die Originalkurve ist **konvex**, der Interpolant aber nicht mehr! Und das hat nichts mit Splines als solchen zu tun, sondern ist ein allgemeines Prinzip:

[68] Im Englischen gibt es das schöne Wort **shape information**, das aber mit „Gestaltinformation“ nur sehr hölzern übersetzt wäre.

Differenzierbare Funktionen können bei Interpolation keine Konvexität der Ausgangsdaten erhalten, man muss sich also entweder von der Differenzierbarkeit oder der Interpolation verabschieden, wenn man Konvexität erhalten will.

Das einfachste Beispiel, das zeigt, daß konvexe differenzierbare Interpolation konvexer Daten unmöglich ist, verwendet die Funktion $f(x) = |x|$ und lediglich die fünf Punkte 0, $\pm\frac{1}{2}$ and ± 1. In der Tat werden wir gleich sehen, daß jeder konvexe C^1-Interpolant auf den beiden Intervallen $[-1, 0]$ und $[0, 1]$ linear sein muss und deswegen an der Stelle 0 eben doch nicht konvex sein kann.

Sei g der Interpolant. Wegen der Konvexität von g haben wir für jedes $x \in [0, 1]$ daß

$$g(x) = g(x\,1 + (1-x)\,0) \le (1-x)\;g(0) + x\,g(1) = (1-x)\;f(0) + x\,f(1) = f(x),$$

und somit $g(x) \le f(x)$, $x \in [0, 1]$. Außerdem ist

$$g'\left(\frac{1}{2}\right) = \lim_{h\to 0^+} \frac{g\left(\frac{1}{2}+h\right) - g\left(\frac{1}{2}\right)}{h} \le \frac{f\left(\frac{1}{2}+h\right) - f\left(\frac{1}{2}\right)}{h} = 1$$

sowie

$$g'\left(\frac{1}{2}\right) = \lim_{h\to 0^+} \frac{g\left(\frac{1}{2}\right) - g\left(\frac{1}{2}-h\right)}{h} \ge \frac{f\left(\frac{1}{2}\right) - f\left(\frac{1}{2}-h\right)}{h} = 1,$$

so daß $g'\left(\frac{1}{2}\right) = 1$. Nehmen wir nun an, es gäbe ein $x^* \in \left[0, \frac{1}{2}\right]$ so daß $g(x^*) < f(x^*)$ und schreiben wir $x \in \left[x^*, \frac{1}{2}\right]$ als

$$x = \lambda x^* + (1-\lambda)\frac{1}{2}, \qquad \lambda = \frac{\frac{1}{2}-x}{\frac{1}{2}-x^*} \in [0, 1],$$

dann erhalten wir, wieder wegen der Konvexität, daß

$$\begin{aligned} g(x) &\le \lambda\, g(x^*) + (1-\lambda)\, g\left(\frac{1}{2}\right) = \lambda\, f(x^*) + (1-\lambda)\, f\left(\frac{1}{2}\right) + \lambda\,(g(x^*) - f(x^*)) \\ &= f\left(\frac{1}{2}\right) - \left(\frac{1}{2} - x\right) + \lambda\,(g(x^*) - f(x^*)) \\ &= x + \frac{\frac{1}{2}-x}{\frac{1}{2}-x^*}\,(g(x^*) - f(x^*)). \end{aligned}$$

Also ergibt sich für $h > 0$,

$$g\left(\frac{1}{2}\right) - g\left(\frac{1}{2}-h\right) \;\ge\; h\left(1 - \frac{g(x^*) - f(x^*)}{\frac{1}{2}-x^*}\right),$$

was zum Widerspruch

$$g'\left(\frac{1}{2}\right) \geq 1 - \frac{g(x^*) - f(x^*)}{\frac{1}{2} - x^*} > 1$$

führt. Auf genau dieselbe Art und Weise impliziert die Existenz eines $x^* \in \left[\frac{1}{2}, 1\right]$ mit $g(x^*) < f(x^*)$ daß $g'\left(\frac{1}{2}\right) < 1$ sein müsste - wieder ein Widerspruch. Damit muss aber $g = f$ sein und daher ist g leider nicht mehr differenzierbar an $x = 0$.

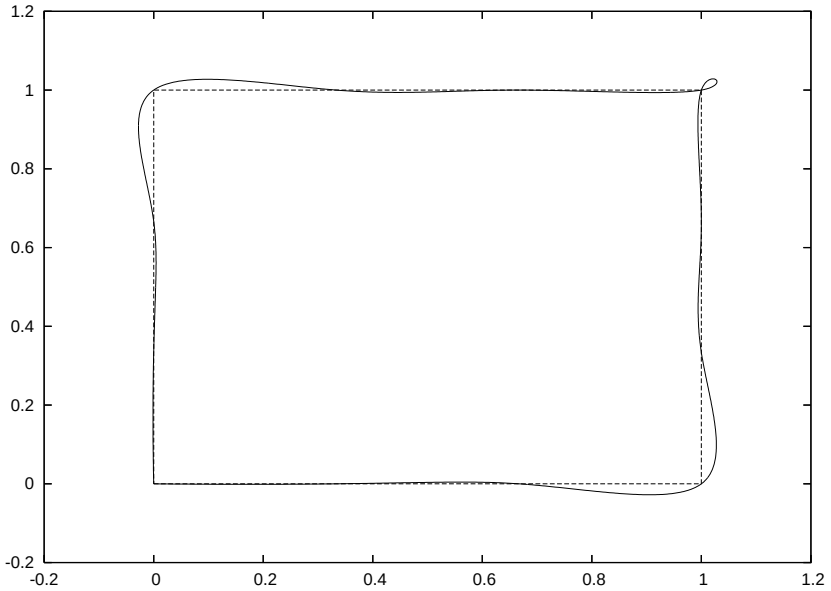


Abbildung 4.7: Eine Interpolationsrunde rund um das Einheitsquadrat zeigt beide möglichen Effekte: Überschießen und Schleifen.

Als „krönenden“ Abschluss dieses Abschnitts gibt Abb. 4.7 noch ein Beispiel, das Interpolation nicht in gutem Licht dastehen lässt.

4.2.3 Die Behandlung von Ecken

Aber was nun tun, wenn wir es mit Daten zu tun haben, die aus sehr aufwendigen Messungen[69] oder Berechnungen stammen und die wir deswegen nicht

[69] In der Koordinatenmeßtechnik wird sehr viel Geld in Hardware zur extrem genauen Messung von Punkten und Abständen investiert.

nur approximieren dürfen, sondern eben wirklich reproduzieren, das heißt interpolieren müssen, ganz einfach weil Abweichungen von diesen Daten nicht toleriert wird? Nun, es gibt einen relativ einfachen Trick, mit Ecken und Kanten in solchen Daten fertigzuwerden:

> Verwende Eckenerkennung und füge für diese Stellen einen m-fachen Knoten ein. Der reproduziert die Ecke und ist außerdem automatisch eine Greville-Abszisse, das heißt, an dieser Stelle wird auch wirklich interpoliert.

Bleibt die Frage, wie man Eckenerkennung durchführt. Aber dafür gibt es jede Menge von Methoden:

- Man bestimmt eine zweite dividierte Differenz der Datenpunkte - sofern man zugehörige Parameterwerte hat. Unter Annahme einer gleichmäßigen Paramterisierung[70] hat man es dann einfach mit $\left\|\mathbf{d}_{j+1} - 2\mathbf{d}_j + \mathbf{d}_{j-1}\right\|$ zu tun. Wenn dieser Wert groß ist[71], dann hat man eine Ecke gefunden.
- Man betrachtet den *Winkel* zwischen zwei aufeinanderfolgenden Datensegmenten:
$$\alpha_j = \arccos \frac{(\mathbf{d}_{j+1} - \mathbf{d}_j)^T (\mathbf{d}_j - \mathbf{d}_{j-1})}{\left\|\mathbf{d}_{j+1} - \mathbf{d}_j\right\| \left\|\mathbf{d}_j - \mathbf{d}_{j-1}\right\|}$$
und nimmt eine Ecke dort an, wo α_j „weit genug“ von π entfernt ist.
- Man verwendet Wavelet-Methoden. Wie in Abb. 4.8 dargestellt, zeigen die Waveletkoeffizienten die Lage der Ecken an, man kann aus ihrer Amplitude und der Art und Weise wie sie ihre Vorzeichen wechseln, unter gewissen Umständen sogar Rückschlüsse auf Art und Winkel der Ecke ziehen.

Eine Kombination zweier Ansätze, dividierte Differenzen und Wavelets, hat in der Tat sehr gute Ergebnisse in der Koordinatenmesstechnik erziehlt (Maresch, 2006). Es lohnt sich allerdings, die vorgegebene Kurve feiner abzutasten um so von äquidistanten Punkten auf einer stückweise linearen Kurve ausgehen zu können.

4.2.4 Warum Approximation besser sein kann

Interpolation hat also ihre Grenzen und diese beruhen, wie wir gesehen haben, auf prinzipiellen Gründen und nicht auf technischen Beschränkungen oder unserer Unfähigkeit. So gesehen kann die Rekonstruktion einer Funktion f aus

[70] Also einer Bogenlängenabtastung, die übrigens nicht ganz einfach zu bestimmen ist.
[71] Was auch immer das konkret bedeutet.

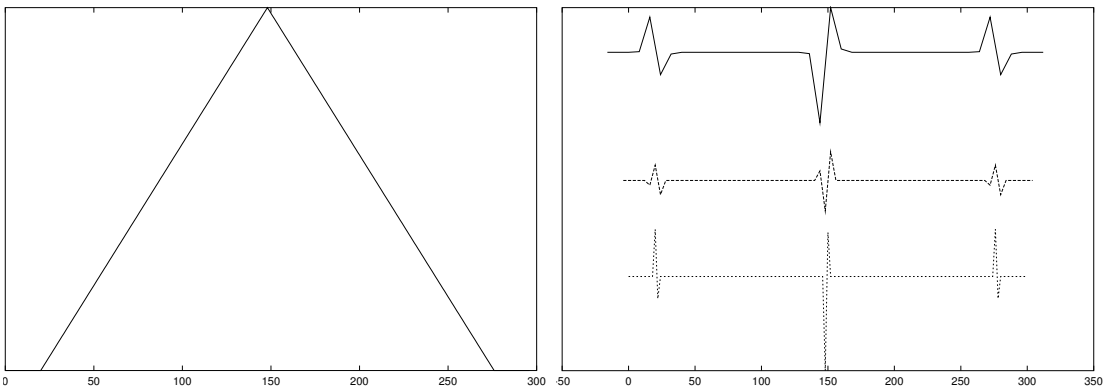


Abbildung 4.8: Die "D_3" Daubechies–Wavelet-Koeffizienten *(rechts)* einer Funktion mit Ecken *(links)*. „Signifikante" Koeffizienten weisen ziemlich deutlich auf die Ecken hin.

Abtastwerten sehr komplex sein und zu Artefakten führen. Einen Teil davon kann man zwar durch Eckenerkennung in den Griff bekommen, aber dennoch bleiben da schon noch einige Details offen ...

Ein vollständig anderer Ansatz zur **Rekonstruktion** von Funktionen aus Abtastwerten ist der vorher schon einmal erwähnte **Schoenbergoperator**

$$\mathscr{S}_m f = \sum_{j=1}^{n} f(x_j)\ N_j^m(\cdot\,|\,T)\,, \qquad x_j = \frac{1}{m}\sum_{k=1}^{m} t_{j+k}, \quad j = 1,\ldots,n, \tag{4.10}$$

ein sogenannter **Quasi-Interpolant** an den **Greville-Abszissen**. Und daß die Greville-Abszissen hier auftauchen, das ist **kein** Zufall!

Proposition 4.3 *Der Schoenbergoperator besitzt* ***lineare Exaktheit****, das heißt,* $\mathscr{S}_m \ell = \ell$ *für* $\ell \in \Pi_1$ *und* $m \geq 1$*.*

Beweis: Im Fall $m = 1$ ist die Proposition offensichtlich korrekt, da wir es dann mit dem stückweise linearen Interpolanten zu einer linearen Funktion zu tun haben, der wieder linear sein muss. Für $m \geq 2$ haben wir hingegen, daß

$$\mathscr{S}_m \ell \;=\; \sum_{j=1}^{n} \ell\left(\frac{1}{m}\sum_{k=1}^{m} t_{j+k}\right) N_j^m(\cdot\,|\,T) = \frac{1}{m}\sum_{j=1}^{n}\sum_{k=1}^{m} \ell(t_{j+k})\ N_j^m(\cdot\,|\,T)\,,$$

und wegen (3.10),

$$\begin{aligned}(\mathscr{S}_m\ell)' &= \sum_{j=1}^{n} \frac{1}{t_{j+m}-t_j} \sum_{k=1}^{m} \left(\ell\left(t_{j+k}\right)-\ell\left(t_{j-1+k}\right)\right) N_j^{m-1}\left(\cdot\,|\,T\right)\\ &= \sum_{j=1}^{n} \frac{\ell\left(t_{j+m}\right)-\ell\left(t_j\right)}{t_{j+m}-t_j} N_j^{m-1}\left(\cdot\,|\,T\right) = \ell',\end{aligned}$$

was eine Konstante ist. Daher ist $\mathscr{S}_m\ell$ aber eine lineare Funktion mit derselben Steigung wie ℓ, und da wenigstens einer der Endknoten die Vielfachheit m oder $m+1$ hat, interpoliert $\mathscr{S}_m\ell$ dort auch noch. Dann bleibt den beiden Funktionen aber wirklich nichts anderes mehr übrig, als identisch zu sein. □

Der oder zumindest ein wichtiger Grund für die Verwendung der Greville-Abszissen ist die lineare Exaktheit des zugehörigen Schoenbergoperators.

Aber warum und wofür ist lineare Exaktheit so erstrebenswert? Ganz einfach: Unter dieser Voraussetzung können wir ein *quantitatives* Resultat für den *globalen* Approximationsfehler angeben, das uns garantiert, daß bei hinreichend feiner Abtastung jede Funktion durch den Schoenbergoperator beliebig gut approximiert werden kann. Wir werden uns auch den Beweis ansehen, denn der ist einerseits ganz illustrativ und der Prototyp für allgemeinere Resultate in Sachen **Approximationsordnung**. Mehr zu dem Thema findet sich in (Boor, 1990; Nürnberger, 1989; Schumaker, 1981).

Satz 4.4 *Für* $f \in C^2\left[t_{m+1}, t_{n+1}\right]$ *gilt*

$$\|f-\mathscr{S}_m f\|_\infty := \max_{x\in[t_{m+1},t_{n+1}]} |f(x)-\mathscr{S}_m f(x)| \le m^2\,\|f''\|_\infty\; h^2, \tag{4.11}$$

wobei

$$h = \max_{j=1,\ldots,m+n} |t_{j+1}-t_j| \tag{4.12}$$

der maximale ***Knotenabstand*** *ist.*

Beweis: Der Beweis ist erstaunlich einfach[72]: Wir betrachten ein nichttriviales Knotenintervall $I_j = [t_j, t_{j+1})$, bemerken, daß

$$\mathscr{S}_m f|_{I_j} = \sum_{k=j-m}^{j} f\left(\frac{t_{k+1}+\cdots+t_{k+m}}{m}\right) N_k^m\left(\cdot|T\right), \tag{4.13}$$

[72] Nur nicht für die, die sowieso alles für klar und trivial halten.

und wählen $t^* = \frac{1}{2}(t_{j-m+1} + t_{j+m})$ als den Mittelpunkt des Intervalls

$$J_j = [t_{j-m+1}, t_{j+m}] \supset I_j,$$

das von den „relevanten" Knoten[73] aufgespannt wird und das alle Greville–Abszissen enthält, die auf der rechte Seite von (4.13) auftauchen können. Eine Taylorentwicklung von f um t^* ergibt, daß

$$f(x) = \underbrace{f(t^*) + (x - t^*) f'(t^*)}_{=:T_1 f} + \frac{(x - t^*)^2}{2} f''(\xi), \qquad \xi \in (x, t^*),$$

und daher, für $x \in J_j$,

$$\begin{aligned} |f(x) - T_1 f(x)| &= \frac{|x - t^*|^2}{2} |f''(\xi)| \le \frac{\frac{1}{4}(t_{j+m} - t_{j-m+1})^2}{2} \max_{x \in I_j} |f''(x)| \\ &\le \frac{m^2 h^2}{2} \|f''\|_\infty, \end{aligned} \tag{4.14}$$

weil $t_{j+m} - t_{j-m+1} \le (2m-1)h$. Insbesondere erhalten wir für $k = j-m, \dots, j$

$$|f - T_1 f|\left(\frac{t_{k+1} + \cdots + t_{k+m}}{m}\right) \le \frac{m^2 h^2}{2} \|f''\|_\infty \tag{4.15}$$

und damit

$$\mathscr{S}_m (f - T_1 f)|_{I_j} \le \frac{m^2 h^2}{2} \|f''\|_\infty \underbrace{\sum_{k=j-m}^{j} N_k^m(\cdot|T)}_{\le 1} \le \frac{m^2 h^2}{2} \|f''\|_\infty .$$

Daraus und aus (4.15) erhalten wir schließlich die finale Abschätzung

$$\begin{aligned} \max_{x \in I_j} |\mathscr{S}_m f - f|(x) &\le \max_{x \in I_j} |\mathscr{S}_m (f - T_1 f)|(x) + \max_{x \in I_j} |T_1 f - f|(x) \\ &\le m^2 h^2 \|f''\|_\infty, \end{aligned}$$

aber da die rechte Seite unabhängig von j ist, gilt die Abschätzung für alle Intervalle *gleichzeitig*, also global, was genau (4.11) ist. □

Bemerkung 4.5 *Wenn man den Beweis etwas genauer ansieht, erkennt man, daß*

$$\max_{x \in I_j} |\mathscr{S}_m f - f|(x) \le m^2 h^2 \max_{x \in J_j} |f''(x)|, \qquad j = m+1, \dots, n, \tag{4.16}$$

ist, was eine lokale *Version des obigen Satzes liefert und zeigt, daß man Knoten dort dicht plazieren muss, wo die Krümmung hoch ist.*

[73] Das sind diejenigen Knoten, die zu B–Splines gehören, die im Intervall I_j ungleich Null sind.

Wir fassen zusammen: Jenseits der technischen Details enthält der Beweis zwei wesentliche Zutaten:

Lokalität: Auf einem bestimmten Knotenintervall ist nur eine kleine Zahl von B-Splines aktiv und der Bereich des Schoenber-Splines, der vom Verhalten von f in diesem Intervall I_j beeinflusst wird, hat höchstens Größe mh.

Polynomerhaltung: Die Greville-Abszissen sind so gewählt, daß der lineare Anteil des Taylorpolynoms bei der Differenz zwischen Funktion und Splineapproximant irrelevant wird, so daß ein Vielfaches von h^2 übrigbleibt, was für $h \to 0$ natürlich auch sehr viel schneller gegen Null geht als h. Das gilt natürlich nur für lineare Exaktheit, könnten wir mehr Polynome reprduzieren, dann würde auch ein größerer Teil des Taylorpolynoms eliminiert werden! Allerdings ist das mit der höheren Ordnung nicht mehr so einfach, man braucht dann auch Ableitungen von f im Quasiinterpolanten ...

Diese Art Argument ist wahrscheinlich viel älter als (Schoenberg, 1973) und funktioniert in viel allgemeineren Sitationen, zum Beispiel bei der Untersuchung von Ganzzahltranslaten von Funktionen mit kompakten Träger im Umfeld von translationsinvarianten Räumen und Wavelets. Das klingt toll und allgemein, aber der Prototyp dort sind auch „nur“ Splines, genauer **kardinale Splines**, also die B-Splines zur unendlichen Knotenfolge $T = h\mathbb{Z}$.

Although this may seen as a parodox, all exact science is dominated by the idea of approximation.

Bertrand Russell

5 Approximation

Wenn also Interpolation nicht so wirklich toll ist und für prinzipielle Schwierigkeiten sorgt, dann sollte man die exakte Reproduktion der Datenpunkte vielleicht gleich aufgeben und dafür mehr Wert auf gute Approximationseigenschaften legen.

5.1 Glättungssplines

Wir haben schon gelernt, daß der natürliche Spline die Lösung eines bestimmten Minimierungsproblems ist, auch wenn es nicht so klar ist, wie sinnvoll es ist, genau dieses Funktional zu minimieren. Aber immerhin - Minimierung scheint durchaus keine schlechte Idee zu sein und deswegen werden wir in diesem Abschnitt Interpolanten oder Approximanten betrachten, die Lösungen von Minimierungsproblem sind. Typischerweise werden solche zu minimierenden Funktionale gewichtete Kombinationen aus einem **Approximationsfunktional** und einem **Glattheitsfunktional** sein, die messen, wie weit der Spline von den vorgegebenen Daten entfernt sein darf beziehungsweise wie gut er sich zu benehmen hat. Dabei können wir natürlich sowohl die Funktionale[74] als auch die verwendete Norm variieren und wir werden in der Tat auch sehen, daß dies zu durchaus unterschiedlichen Resultaten führen kann

5.1.1 Interpolation und Minimierung

Die erste, einfachste und unmittelbarste Idee ist natürlich, unter allen verfügbaren[75] Interpolanten denjenigen auszuwählen, der ein vorgegebenes Funktional

[74] Wollen wir beispielsweise nur vorgegebene Werte approximieren oder vielleicht auch Ableitungen und wollen wir „Glattheit“ eher über eine erste, zweite, dritte oder siebenundzwanzigste Ableitung beschreiben?

[75] Diese Menge sollte allerdings schon noch etwas eingeschränkt sein, endlichdimensionale Räume könnten da gewiss nicht schaden.

minimiert - dieses Konzept der **Optimalinterpolation**[76] ist auch im statistischen Kontext sehr beliebt[77] und dort unter dem Namen **Kriging** bekannt, siehe z.B. (Christensen, 2001).

Schauen wir uns doch einmal zwei Beispiele solcher Interpolationsprozesse mit Splines an. Dazu wählen wir Interpolationsstellen $X = \{x_j : j = 1, \ldots, n'\}$, $n' \leq n$, und erhalten so ein *unterbestimmtes* Interpolationsproblem, also auch ein unterbestimmtes lineares Gleichungssystem, und machen die Lösung dadurch eindeutig, daß wir ein zusätzliches lineares Funktional minimieren - im statistischen Umfeld der linearen Modelle wäre das eben dann ein Kovarianzschätzer[78]. Konkret könnte unser Minimierungsproblem die Gestalt

$$\min_{f \in S_m(T)} \int_{\mathbb{R}} |f''(x)|^2 \, dx, \qquad f(X) = \mathbf{y}, \quad \mathbf{y} \in \mathbb{R}^X, \tag{5.1}$$

haben. Und wieder ist die Matrix-Vektor-Notation sehr hilfreich. Unter Verwendung der Splinedarstellung $f = S_m\mathbf{d} = \mathbf{d}\mathbf{N}_m$ können wir das Minimierungsproblem (5.1) bezüglich des (Zeilen-) Vektors $\mathbf{d}$ als

$$\min_{\mathbf{d}} \int_{\mathbb{R}} \mathbf{d}\mathbf{N}_m''(x)\mathbf{N}_m''(x)^T\mathbf{d} \, dx =: \mathbf{d}\mathbf{A}\mathbf{d}^T, \qquad \mathbf{d}\mathbf{N}_m(X) = \mathbf{y}, \tag{5.2}$$

schreiben, was ein quadratisches Optimierungsproblem mit Gleichheitsrandbedingungen ist. Mittels der **Lagrangemultiplikatoren**[79], siehe(Sauer, 2002b), ergibt sich das Minimum dann ganz einfach[80] als Lösung des linearen Gleichungssystems

$$\begin{bmatrix} 2\mathbf{A} & \mathbf{N}_m(X) \\ \mathbf{N}_m^T(X) & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{d}^T \\ \lambda \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{y}^T \end{bmatrix}, \tag{5.3}$$

wobei, gemäß (3.12),

$$\begin{aligned} \mathbf{A} &= \left[\int_{\mathbb{R}} N_j^m(x)'' \, N_k^m(x)'' \, dx \, : \, j,k = 1, \ldots, n \right] \\ &= \int_{\mathbb{R}} \widehat{\mathbf{G}}_2 \mathbf{N}_m(x)\mathbf{N}_m^T(x)\widehat{\mathbf{G}}_2^T \, dx = \widehat{\mathbf{G}}_2 \left(\int_{\mathbb{R}} \mathbf{N}_m(x)\mathbf{N}_m^T(x) \, dx \right) \widehat{\mathbf{G}}_2^T \end{aligned}$$

[76] Siehe beispielsweise die Literaturverweise in (Micchelli, 1986).

[77] Dann ist das Funktoinal normalerweise die Varianz der Zufallsvariable.

[78] Man kann sehr viel Mathematik so in statistischer Terminologie umformulieren, daß sie kein normaler Mathematiker mehr versteht.

[79] Optimierer würden das als *Kuhn-Tucker-Bedingungen* bezeichnen und möglicherweise die Namensliste sogar noch deutlich verlängern.

[80] Man muss sich natürlich noch überlegen, daß es da genau ein Minimum geben muss und daß das einzige Extremum unter diesen Nebenbedingungen auch wirklich ein Minimum ist - aber anschaulich sucht man ja hier „nur“ nach dem Minimum einer nach oben geöffneten Parabel.

eine quadratische, positiv definite Matrix ist. Außerdem ist die **Gram-Matrix** der B-Splines

$$\mathbf{B}_m = \mathbf{B}_m(T) := \int_{\mathbb{R}} \mathbf{N}_m(x)\mathbf{N}_m^T(x)\,dx \tag{5.4}$$

ja sogar *strikt* positiv definit und daher gilt $\mathbf{Ax} = 0$ genau dann, wenn $\mathbf{G}_2\mathbf{x} = 0$ ist, also genau dann, wenn $\mathbf{x}$ Segment einer linearen Folge ist. Daher hat das lineare Gleichungssystem in (5.3) genau dann eine *eindeutige* Lösung wenn das Interpolationsproblem nicht durch eine lineare Funktion gelöst werden kann, das heißt, wenn die Datenpunkte nicht auf einer Geraden liegen.

Dieser Ansatz der Minimalinterpolation funktioniert immer, solange man nur ein **quadratisches Funktional** zu minimieren versucht und führt immer zu einem linearen Gleichungssystem. Kompliziertere Funktionale würden dann zu *nichtlinearen* Gleichungen führen, die auch wesentlich komplexere Lösungsmethoden erforderlich machen würden.

Andere Minimierungsfunktionale kann man durch *Diskretisierung* des Energiefunktionals[81], das heißt, durch Berechnung des Vektors

$$\mathbf{f} := f''(Z) = \mathbf{d}\mathbf{N}_m(Z)'', \qquad Z \subset \mathbb{R},$$

woraufhin man eine Vektornorm von $\mathbf{f}$, beispielsweise

$$\|\mathbf{f}\|_1 = \sum_{z\in Z} |f''(z)|$$

minimiert. Unter der Annahme $\#Z = N$, das heißt, $Z = \{z_1, \ldots, z_N\}$, ergäbe sich das Minimierungsproblem[82] dann als

$$\min_{\mathbf{d},\mathbf{u}} \mathbf{1}^T\mathbf{u} = \sum_{j=1}^{N} u_j, \qquad \begin{bmatrix} \mathbf{N}_m^T(X) & \mathbf{0} \\ \mathbf{N}_m^T(Z)'' & -\mathbf{I} \\ -\mathbf{N}_m^T(Z)'' & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{d}^T \\ \mathbf{u} \end{bmatrix} \begin{bmatrix} = \\ \leq \\ \leq \end{bmatrix} \begin{bmatrix} \mathbf{y}^T \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix},$$

und kann beispielsweise mit linearer Programmierung gelöst werden. Minimierung bezüglich der ℓ_1-Norm ist ein gebräuchliches Prinzip bei der Bildverarbeitung und den dabei entstehendend inversen Problemen, siehe (Mallat, 1999; Sauer, 2003).

[81] Wir werden „Energiefunktional" als Synonym für „Glattheitsfunktional" verwenden, die anschauliche Idee dahinter sollte ja inzwischen klar sein.

[82] Wie man darauf kommt? Keine Angst, das werden wir gleich sehen, und zwar mehr als nur einmal.

5.1.2 Minimierung der Supremumsnorm und lineare Optimierung

Anstatt „mittlere“ Eigenschaften einer gewissen Ableitung zu minimieren, können wir auch Verfahren verwenden, die darauf abzielen, ein **globales Maximum** dieser Ableitung zu minimeren, das heißt, man betrachtet

$$\min_{\mathbf{d}} \left\| S_m^{(k)}\mathbf{d} \right\|_\infty = \min_{\mathbf{d}} \max_{t\in[t_{m+1},t_{n+1}]} \left| S_m^{(k)}\mathbf{d}(t) \right|, \qquad \mathbf{d}\mathbf{N}_m(X) = \mathbf{y},$$

wobei unsere Randbedingung immer noch so gewählt ist, daß die Splinekurve interpoliert, also immer noch ein **interpolierender Spline** vorliegt - wir verändern hier erst einmal nur das zu minimierende Funktional. Sehen wir uns dieses Problem erst einmal für den Fall $k = m-1$, also die höchste Differenzierbarkeitsordnung[83] an, denn dann ist die Splinekurve

$$S_m^{(m-1)}\mathbf{d} = \mathbf{d}\,\mathbf{N}_m^{(m-1)} = \mathbf{d}\widehat{\mathbf{G}}_{m-1}\,\mathbf{N}_1\left(\cdot \,|\, \widehat{T}\right)$$

eine **stückweise lineare Kurve**, die ihr Maximum an einem der Knoten annehmen muss und der Wert dieses Maximums ist die Norm des entsprechenden Kontrollpunkts. Als „Koeffizientennorm“ auf $\mathbb{R}^d$ wählen wir ebenfalls die ∞-Norm und definieren

$$\nu := \max_{t\in[t_{m+1},t_{n+1}]} \left\| S_m^{(m-1)}\mathbf{d} \right\|_\infty = \max_{j=1,\dots,n-m+1} \left\| \left(\mathbf{d}\widehat{\mathbf{G}}_{m-1}\right)_j \right\|_\infty .$$

Mit anderen Worten: ν ist die kleinste positive Zahl so daß

$$\left| \left(\mathbf{d}\widehat{\mathbf{G}}_{m-1}\right)_j \right| \le \nu\mathbf{1}, \qquad j = 1,\dots,n-m+1,$$

das heißt,

$$-\nu\mathbf{1} \le \mathbf{d}\widehat{\mathbf{G}}_{m-1} \le \nu\mathbf{1}$$

oder

$$\pm\left(\mathbf{d}\widehat{\mathbf{G}}_{m-1}\right)_j \le \nu\mathbf{1}, \qquad j = 1,\dots,n-m+1, \tag{5.5}$$

so daß wir die Ungleichungen aus (5.5) in der Matrixform

$$\pm\mathbf{d}\widehat{\mathbf{G}}_{m-1} - \underbrace{\mathbf{1}_d\,\mathbf{1}_{n-m+1}}_{=:\mathbf{1}_{d\times n+m-1}}\nu \le \mathbf{0}_{d\times n+m-1}$$

zusammenfassen können, die wir dann nur noch um die linearen Gleichheitsbedingungen aus der Interpolation,

$$\mathbf{d}\,\mathbf{N}_m(X) = \mathbf{y},$$

[83] Für $m = 3$ bedeutet das gerade, den größten Wert von $S_m''\mathbf{d}$ zu minimieren.

erweitern müssen. Schließlich transponieren wir das Ganze noch und stellen fest, daß die Lösung des linearen Programms

$$\min_{\mathbf{d},\nu} \nu, \qquad \begin{bmatrix} \mathbf{N}_m^T(X) & \mathbf{0} \\ \widehat{\mathbf{G}}_{m-1}^T & -\mathbf{1}_{n-m+1\times d} \\ -\widehat{\mathbf{G}}_{m-1}^T & -\mathbf{1}_{n-m+1\times d} \end{bmatrix} \begin{bmatrix} \mathbf{d}^T \\ \nu \end{bmatrix} \begin{bmatrix} = \\ \leq \\ \leq \end{bmatrix} \begin{bmatrix} \mathbf{y}^T \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix} \tag{5.7}$$

uns dasjenige Kontrollpolygon liefert, für das der zugehörige Spline minimalen *globalen* Absolutbetrag der $m+1$-ten Ableitung hat. Ein paar Bemerkungen sind aber an dieser Stelle schon nötig:

1. Die meisten Implementierungen des Simplexalgorithmus oder generell von Lösungsverfahren für lineare Programmierung kennen nur *vektorwertige* rechte Seiten in (5.7), nicht aber *matrixwertige*, so wie sie dort im Moment stehen. Das ist aber kein wirkliches Problem, man muss nur die Spalten der Matrix übereinanderstapeln. In `octave` wird diese Aufgabe vom Befehl `reshape` ausgeführt, mathematisch ist das als der „vec"-Operator, siehe z.B. (Marcus & Minc, 1969), bekannt.

2. Die (5.7) verlangt einem Lösungsverfahren für lineare Optimierung schon einiges ab! Der Grund ist, daß die Variablen $\mathbf{d}$ normalerweise auch negative Werte annehmen dürfen[84] und auch sonst nicht direkt beschränkt sind, sondern nur indirekt durch die Nebenbedingung, die sie mit der zu minimierenden Variable verknüpft. In der Optimierung bezeichnet man dies als eine **freie Variable** und die müssen erkannt und geeignet behandelt[85] werden. Eine naive Implementierung des Simplexalgorithmus wie in (Sauer, 2002b) würde an diesem Optimierungsproblem scheitern.

3. Viel wichtiger ist es, zu beachten, daß (5.7) **nicht** komponentenweise betrachtet werden kann, denn die Komponenten der Koeffizienten $\mathbf{d}_j$, $j = 1, \ldots, n$, sind ja über die Variable ν gekoppelt, die das Maximum über **alle** Komponenten **aller** Kontrollpunkte ist.

4. Würden wir jede Komponente individuell durch ν_j beschränken und dann über $\nu_1 + \cdots + \nu_d$ minimieren, dann entspräche die der Verwendung der 1-Norm als Koeffizientennorm im $\mathbb{R}^d$.

5. Im allgemeine führt das Miniminierungsproblem (5.7) allerdings schon zu Optimierungsproblemen mit $dn + 1$ Variablen, was für manache realistischen Werte von n schon recht groß werden kann. Man sollte außerdem

[84] Es gibt ja keinen Grund, dies a priori auszuschließen.
[85] Genauer gesagt ausgetauscht.

schon bedenken, daß die Komplexität des Simplexalgorithmus für n Variablen bei $O(2^n)$ liegt, auch wenn das nur bei sehr speziellen und eher akademischen Problemen passiert, siehe z.B. (Sauer, 2002b), und im Mittel der Erfahrungswert wohl eher bei $O(n)$ zu finden ist, siehe (Nocedal & Wright, 1999; Spellucci, 1993). Aber garantiert ist halt nichts ...

Aber was passiert im Fall $k < m-1$ von Ableitungen niedrigerer Ordnung? Hier werden die Maxima ja nicht mehr an den Knoten angenommen, obwohl natürlich die Idee von (5.7) immer noch funktioniert - wir müssen ja nur $\widehat{\mathbf{G}}_{m-1}$ durch $\widehat{\mathbf{G}}_k$ ersetzen. Was wir dann minimieren ist natürlich nicht mehr die Norm der Ableitung, sondern nur die Norm des größten Koeffizienten der Ableitung. Wegen

$$\left\| S_m \mathbf{d}(x) \right\| \le \sum_{j=1}^{n} \left\| \mathbf{d}_j \right\| N_j^m(x \mid T) \le \max_{j=1,\ldots,n} \left\| \mathbf{d}_j \right\| \underbrace{\sum_{j=1}^{n} N_j^m(x \mid T)}_{=1} = \max_{j=1,\ldots,n} \left\| \mathbf{d}_j \right\|,$$

führen „global kleine“ Kontrollpunkte aber auch zu „global kleinen“ Kurven und somit ist die zu minimierende Zielfunktion schon immer noch sinnvoll. Und vor allem sind halt auch effiziente Methoden verfügbar, um dieses Minimimierungsproblem zu lösen, beispielsweise `lp_solve` oder `glpk`, siehe Abb. A.1.

Wie man solche Splines dann auch effizient berechnet, insbesondere die Komponenten der Matrizen in (5.3) und (5.7), damit werden wir uns später noch eingehend befassen.

5.1.3 Ein erster Vergleich

Bevor wir mit der Theorie weitermachen, wollen wir uns nun doch mal anhand eines Beispiels ansehen, was diese beiden Ansätze der minimierenden Interpolation denn nun so zu leisten imstande sind und vor allem, wo sie sich unterscheiden. Dazu legen wir zuerst einmal die Knotenfolge

```
octave> T = [ 0 0 0 (0:10) 10 10 10 ];
```

und die Interpolationsbedingungen

```
octave> X = (0:10); y = [ 0 0 0 0 0 1 0 0 0 0 0 ];
```

fest. Für den Spline mit minimaler 2-Norm benötigen wir außerdem die Gram-Matrix der B-Splines, das ist die Matrix $\mathbf{A}$ aus (5.3); wie diese bestimmt wird, das werden wir in Abschnitt 5.1.5 noch im Detail ausarbeiten, für den Moment reicht es uns, zu wissen, daß es dafür eine Funktion `BSplGramDer` gibt:

```
octave> A = BSplGramDer( 3,T,2 ); V = BSplVander( 3,T,X );
```

Da wir die Vandermondematrix gleich mitberechnet haben, können wir auch schon unsere Matrix und die rechte Seite zusammensetzen und den Koeffizientenvektor bestimmen:

```
octave> M = [ 2*A, V ; V', zeros( 11 ) ]; b = [ zeros( 13,1 ); y' ];
octave> d = ( M\b )(1:13)';
```

Schließlich beginnen wir schon einmal, die Vorgaben und den tatsächlich interpolierenden Spline zu plotten:

```
octave> plot( (0:10),y,"*" ); hold on;
octave> Z = linspace( 0,10,1000 ); plot( Z,d*BSplVander( 3,T,Z ), 'r' );
```

So, nun aber zur ∞-Norm. Dazu benötigen wir die folgende Kombination aus Matrix und rechter Seite

```
octave> G = BSplDerMat( 3,T,2 );
octave> M = [ V', zeros( 11,1 ); G', -ones( 11,1 ); -G', -ones( 11,1 ) ];
octave> b = [ y'; zeros( 22,1 ) ];
```

sowie die Vektoren für Zielfunktion und Ungleichungsbedingungen

```
octave> f = [ zeros( 13,1 ); 1 ]; e = [ zeros( 11,1 ); -ones( 22,1 ) ];
```

und schon können wir uns an unser lineares Programm machen:

```
octave> [o,x,du] = lp_solve ( -f,M,b,e,[ -Inf*ones( 13,1 ); 1 ] );
3 matrix contains zero-valued coefficients.
3 matrix contains zero-valued coefficients.
octave> d = x(1:13)';
```

Die Warnung können wir ignorieren[86] und direkt mit dem Plotten unseres Ergebnisses weitermachen:

```
octave> plot( Z,d*BSplVander( 3,T,Z ), 'g' );
```

Wie man deutlich in Abb. 5.1 (links) sieht, ist in diesem Fall das Ergebnis der quadratischen Optimierung deutlich besser als das Ergebnis bei Optimierung der Supremumsnorm, obwohl der Unterschied in der zweiten Ableitung nicht einmal sichtbar ist, wenn man die entsprechenden Funktionen plottet. Es ist aber relativ einfach zu erklären, *was* da passiert: Die Supremumsnorm minimiert das *globale* Maximum der zweiten Ableitung und erreicht so einen Wert von

[86] Ehrlich gesagt verstehe ich sie auch nicht ganz.

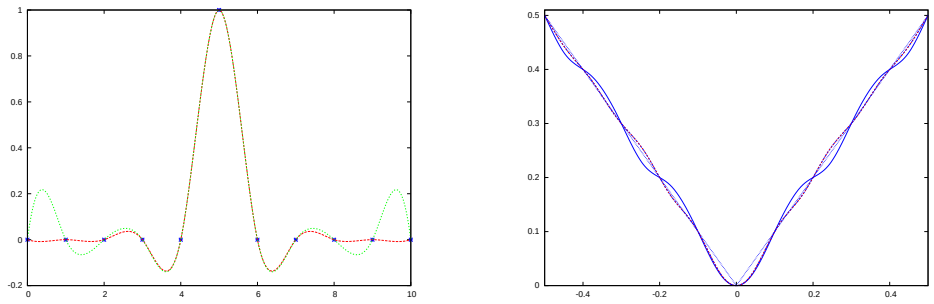


Abbildung 5.1: Ein erster Vergleich zwischen quadratischer Optimierung und Optimierung der Supremumsnorm. Der Interpolationsspline zur Supremumsnorm schwingt deutlich stärker, und zwar in beiden Fällen.

4.3802, der aber dafür an vielen Stellen angenommen werden muß[87]. Auf der anderen Seite ist das quadratische Mittel ist in diesem Fall nur unwesentlich schlechter, nämlich 4.3929, nimmt aber diesen Maximalwert halt nur an einer Stelle, höchstwahrscheinlich an dem „Ausreißer“ in der Mitte, an und oszillliert außen wesentlich schwächer.

Das zweite Beispiel ist die Interpolation der Betragsfunktion $f(x) = |x|$, $x \in [-1, 1]$ mit 21 gleichverteilten Interpolationspunkten an den Stellen $\pm\frac{k}{10}$, $k = 0, \ldots, 10$ und hinreichen vielen ebenfalls gleichverteilten Knoten:

```
octave> X = ( -1:.1:1 ); y = abs(X);
octave> T = [ -1 -1 -1 linspace( -1,1,30 ) 1 1 1 ];
```

Für die Berechnung nehmen wir nun zwei fertige `octave`-Routinen

```
octave> d1 = OptInt2( 3,T,X,y ); d2 = OptIntInf( 3,T,X,y );
```

und sehen uns das Ergebnis nur auf dem Intervall $\left[-\frac{1}{2}, \frac{1}{2}\right]$ an:

```
octave> Z = linspace( -.5,.5,1000 ); plot( Z,abs(Z),'b' );
octave> plot( Z,d1*BSplVander( 3,T,Z ), 'r' );
octave> plot( Z,d1*BSplVander( 3,T,Z ), 'b' );
```

und auch hier wackelt der Supremumsspline stärker als der quadratische optimale Splineinterpolant, siehe nochmals Abb. 5.1 (diesmal rechts).

Für Interpolation ist wohl also die 2-Norm besser als die ∞-Norm, denn da sind die „Krümmungen“ ja sehr stark durch die diskreten Daten vorgegeben

[87] Das hat etwas mit den Oszillationseigenschaften gleichmäßiger Bestapproximationen zu tun, siehe z.B. (Lorentz, 1966; Sauer, 2002a).

und die ∞-Norm minimiert diese um den Preis *vieler* Stellen mit maximaler Krümmung. Und diesen Preis bezahlt man natürlich besonders stark bei Interpolation: Wenn man die Kurven nicht „schneiden“ darf, sondern die Punkte passieren muss, dann legen die natürlich die Krümmung schon ziemlich stark fest. Trotzdem war das Kapitel nicht umsonst, denn wir verstehen jetzt wenigstens, wie wir diese Probleme aufstellen und behandeln können.

5.1.4 Kleinste Quadrate und Energiefunktionale

Der Minimierungsansatz aus dem vorhergegangenen Abschnitt beruhte immer noch auf *Interpolation*. Natürlich gibt es Situationen, in denen Interpolation der richtige oder einzig mögliche Ansatz ist, der den Daten gerecht wird, aber in vielen Anwendungssituationen hat man es (mindestens) mit einer der beiden folgenden Situationen zu tun:

1. Die Daten sind mit „verrauscht“, also durch unberechnbare Störungen kontaminiert, und der Versuch des Interpolanten, dieses Rauschen zu reproduzieren, führt zu unnötiger und auch unerwünschter Oszillation des Splines.
2. Interpolation ist möglicherweise noch nicht einmal nötig und die Aufgabe besteht, wie beispielsweise beim Fräsen, „nur“ darin, die vorgegebenen Daten mit ausreichender Genauigkeit zu reproduzieren. Und diese Toleranz kann man sogar dazu ausnutzen, „glattere“ Lösungen des Approximationsproblems zu erhalten, die vielleicht sogar eventuell vorhandenes Rauschen verringern.

Der **Glättungsspline** bzw. **smoothing Spline** ist als Lösung des Optimierungsproblems

$$\min_{\mathbf{f}\in\mathbb{S}_m(T)} \sum_{x\in X} \left\|\mathbf{f}(x) - \mathbf{y}_x\right\|_2^2 + \lambda \int_{\mathbb{R}} \left\|\mathbf{f}''(t)\right\|_2^2 \, dt \tag{5.8}$$

definiert, bei dem der Parameter $\lambda > 0$ passend zu wählen ist.

Die Idee des Glättungssplines besteht darin, eine gute Balance zwischen Datentreue ($\lambda = 0$) und Glattheit ($\lambda \to \infty$) im geometrischen Sinn von Nichtoszillation mit Hilfe des Parameters λ zu finden.

Natürlich kann man in (5.8) eine ganze Menge Dinge variieren:

1. Man könnte ein anderes **Glattheitsmaß** als $|\cdot|_2$, das Integral über das Quadrat der zweiten Ableitung, verwenden, beispielsweise Integrale über

Ableitungen höherer Ordnung oder sogar Linearkombinationen[88] von Ableitungen verschiedener Ordnung.

2. Beide Normen, sowohl die im **Approximationsterm** links als auch die im **Glattheitsterm** rechts kann man auch noch mit lokalen Gewichten versehen, die beispielsweise an manchen Stellen höhere Datentreue fordern als an anderen.

Eine ziemlich allgemeine Form des Glätungssplines läßt sich mit **Gewichten** $w_x \geq 0$, $x \in X$, und Parametern[89] $\lambda_1, \ldots, \lambda_{m-1}$ als

$$\min_{f \in S_m(T)} \sum_{x \in X} w_x |f(x) - y_x|^2 + \sum_{r=1}^{m-1} \lambda_r \int_{\mathbb{R}} \left| f^{(r)}(x) \right|^2 \, dx. \tag{5.9}$$

formulieren. Um die Lösung solch eines Problems zu bestimmen, definieren wir zuerst einmal die (Zeilen-) Vektoren

$$f(X) = [f(x) \, : \, x \in X] = \mathbf{d}\,\mathbf{N}_m(X) \qquad \text{sowie} \qquad \mathbf{y} = [y_x \, : \, x \in X]$$

und bemerken, daß mit $\mathbf{W} = \operatorname{diag}[w_x \, : \, x \in X]$ die Beziehung

$$\begin{aligned} \sum_{x \in X} w_x |f(x) - y_x|^2 &= (\mathbf{d}\mathbf{N}_m(X) - \mathbf{y})\,\mathbf{W}\,(\mathbf{d}\mathbf{N}_m(X) - \mathbf{y})^T \\ &= \mathbf{d}\mathbf{N}_m(X)\mathbf{W}\mathbf{N}_m^T(X)\mathbf{d}^T - 2\,\mathbf{d}\mathbf{N}_m(X)\mathbf{W}\mathbf{y}^T + \mathbf{y}^T\mathbf{W}\mathbf{y} \end{aligned}$$

gilt und daß außerdem, wenn wir uns an (5.4) erinneren, auch

$$\int_{\mathbb{R}} \left| f^{(r)}(x) \right|^2 \, dx = \mathbf{d}\mathbf{G}_r\mathbf{B}_{m-r}\mathbf{G}_r^T\mathbf{d}^T$$

erfüllt ist. Also hat das Funktional, das bezüglich $\mathbf{d}$ zu minimieren ist, die Gestalt

$$\Phi_\lambda(\mathbf{d}) = \mathbf{d}\left(\mathbf{N}_m(X)\mathbf{W}\mathbf{N}_m^T(X) + \sum_{r=1}^{m-1} \mathbf{G}_r\mathbf{B}_{m-r}\mathbf{G}_r^T\right)\mathbf{d}^T - 2\,\mathbf{d}\mathbf{N}_m(X)\mathbf{W}\mathbf{y}^T + \mathbf{y}^T\mathbf{W}\mathbf{y},$$

was eine **quadratische Form** mit positiv semindefinitem[90] quadratischem Koeffizienten ist, deren Minimum durch Lösen von $\nabla_{\mathbf{d}}\Phi_\lambda(\mathbf{d}) = 0$ lokalisiert werden kann, also durch Lösen des linearen Gleichungssystems

$$\left(\mathbf{N}_m(X)\mathbf{W}\mathbf{N}_m^T(X) + \sum_{r=1}^{m-1} \lambda_r\mathbf{G}_r\mathbf{B}_{m-r}\mathbf{G}_r^T\right)\mathbf{d}^T = \mathbf{N}_m(X)\mathbf{W}\mathbf{y}^T. \tag{5.10}$$

[88] Wenn es sein muss sogar gewichtete Linearkombinationen!

[89] Warum man höchstens $m-1$ derartige Terme hat? Naja, sie entsprechen ja Ableitungen und ein Spline der Ordnung m hat höchstens $m-1$ stetige Ableitungen.

[90] Und in den meisten Fällen auch strikt positiv definitem.

Wie wir später sehen werden, haben diese **Normalengleichungen** für Splines eine besonders schöne Form - die Matrizen auf der rechten Seite sind bandiert und damit mit einem linearen Aufwand von $O(n)$ lösbar.

Man beachte, daß es für Glättungssplines ausdrücklich *keine* Beziehung zwischen der Dimension des Splineraums und der Anzahl der „Interpolationsbedingungen" gefordert wird. Allerdings sind sie natürlich nicht völlig unabhängig voneinander und beeinflussen schon das Ergebnis:

1. Ist die Dimension des Splineraums zu klein, dann kann dies zu einem ernsthaften Defekt bei der Approximation führen, denn wenn man nicht interpolieren kann, dann gibt es ja einen unvermeidbaren Abstand an den Datenpunkten. Ist nun λ relativ klein gewählt, dann dominiert der Approximationsterm den Glättungsterm möglicherweise sehr stark und die Funktion macht in Sachen Glätte keine großen Fortschritte.

2. Ist hingegen der Splineraum „groß genug" oder möglicherweise sogar „zu groß", dann wird sich für kleine Werte von λ immer ein Fastinterpolant ergeben, der mehr oder weniger durch die vorgegebenen Datenpunkte verläuft.

Auch hier haben wir natürlich wieder eine Menge von Freiheiten, angefangen mit der Wahl der Dimension des Splineraums, über die Lage der Interpolationsstellen X, bis hin zur Wahl des Parameters λ. Für letzteres gibt es das Konzept der **Kreuzvalidierung** (Craven & Wahba, 1979; Golub *et al.*, 1979), das diesen Parameter *automatisch* so bestimmt, daß er in einem gewissen statistischen Sinn optimal ist.

In **Fittingproblemen**, wo ein „glattestmöglicher" Spline gefunden werden muss, der „nahe genug" bei vorgegebenen Daten verläuft, kann man eine andere Strategie verfolgen:

1. Man wählt den Splineraum so groß, daß Interpolation an den Datenpunkten möglich ist[91].

2. Man löst das Problem für $\lambda = 0$, was einen Interpolanten liefert, dessen Koeffizienten sogar $\Phi_0(\mathbf{d}) = 0$ erfüllen. Welchen Interpolanten man hier wählt, ist eigentlich egal[92]

[91] Es ist kein Fehler, daß das Wort „eindeutig" hier nicht auftaucht, Hauptsache es gibt einen Interpolanten, wenn es mehrere gibt - umso besser.

[92] OK, eigentlich braucht man ihn noch nicht einmal zu berechnen, kann gleich mit einem kleinen Wert von λ anfangen.

3. Man vergrößert λ so lange die Approximationsforderung noch erfüllt ist.

Diese Prozedur liefert uns einen Parameter λ für eine „glatteste“ Approximation der vorgegebenen Daten innerhalb einer vorgegebenen Toleranz und basiert trotzdem nur auf der banalen Beobachtung, daß $\Phi_\lambda(\mathbf{d})$ stetig in λ und $\mathbf{d}$ ist und daß

$$\lim_{\lambda \to 0} \Phi_\lambda(\mathbf{d}) = 0$$

ist.

5.1.5 Effiziente Implementierung von Gram-Matrizen

Es wird mal wieder Zeit für ein „technisches“ Kapitelchen, denn jetzt besteht ja wirklich die Notwendigkeit, die **Gram-Matrix** der B-Splines

$$\mathbf{B}_m(T) = \left[\int_{\mathbb{R}} N_j^m(x)\, N_k^m(x)\, dx \;:\; \begin{matrix} j = 1, \ldots, n \\ k = 1, \ldots, n \end{matrix} \right] \tag{5.11}$$

zu berechnen und das sollten wir besser auch auf effiziente Art und Weise machen. Zuerst bemerken wir, daß die Matrix $\mathbf{B}_m(T)$ offensichtlich symmetrisch ist und daß es somit genügt, die Einträge des Teils „oben rechts“ zu berechnen, also die Einträge mit Indizes $j \le k$. Nachdem N_j^m als Träger ja $[t_j, t_{j+m+1}]$ hat und enstprechend N_k^m das Intervall $[t_k, t_{k+m+1}]$, ist das Integral einer Komponente von (5.11) genau dann von Null verschieden, wenn $t_k \in [t_j, t_{j+m}]$, also genau dann, wenn $k \le j + m$ gilt. Anders gesagt, wir müssen für vorgegebenes $j \in \{1, \ldots, n\}$ genau die Integrale

$$\int_{\mathbb{R}} N_j^m(x)\, N_k^m(x)\, dx = \int_{t_j}^{t_{j+m+1}} N_j^m(x)\, N_k^m(x)\, dx, \qquad k = j, \ldots, j + m$$

berechnen und das liefert uns bereits eine wichtige, aber nicht allzu überraschende Beobachtung:

> Die Gram-Matrix $\mathbf{B}_m(T)$ ist eine symmetrische bandierte Matrix mit m Sub- und Superdiagonalen.

Diese Methode ist in der Funktion `BSplGram` implementiert. Um die Gram-Matrix einer Ableitung, das heißt, die Matrix

$$\mathbf{B}_m^{(r)} := \int_{\mathbb{R}} \mathbf{N}_m^{(r)}(x) \mathbf{N}_m^{(r)}(x)^T\, dx = \left[\int_{\mathbb{R}} N_j^m(x)^{(r)}\, N_k^m(x)^{(r)}\, dx \;:\; \begin{matrix} j = 1, \ldots, n \\ k = 1, \ldots, n \end{matrix} \right],$$

zu erhalten, substituieren wir (3.12) und erhalten

$$\begin{aligned}\mathbf{B}_m^{(r)} &= \int_{\mathbb{R}} \widehat{\mathbf{G}}_r \mathbf{N}_{m-r}\left(x \,|\, \widehat{T}_k\right) \mathbf{N}_{m-r}\left(x \,|\, \widehat{T}_k\right)^T \widehat{\mathbf{G}}_r^T \, dx \\ &= \widehat{\mathbf{G}}_r \left(\int_{\mathbb{R}} \mathbf{N}_{m-r}\left(x \,|\, \widehat{T}_k\right) \mathbf{N}_{m-r}\left(x \,|\, \widehat{T}_k\right)^T dx \right) \widehat{\mathbf{G}}_r^T = \widehat{\mathbf{G}}_r \, \mathbf{B}_{m-r}\left(\widehat{T}_k\right) \widehat{\mathbf{G}}_r^T,\end{aligned}$$

was der ökonomischste Weg ist, diese Matrizen zu berechnen - zumal wir ja schon eine Funktion haben, die uns die Matrizen $\widehat{\mathbf{G}}_r$ bestimmt. Es ist außerdem bemerkenswert, daß $\mathbf{B}_{m-r}\left(\widehat{T}_k\right) \in \mathbb{R}^{n-r \times n-r}$ was sehr schön den Rangverlust wiederspiegelt, der ja beim Übergang zu Ableitungen auftritt.

5.1.6 Ein Beispiel eines Glättungssplines

Zuerst sehen wir uns einmal den „Standardfall“ an, in dem Glattheit kubischer Splines als Eigenschaft der zweiten Ableitung angesehen wird. Als Approximationsstellen verwenden wir die Greville-Abszissen bezüglich T. Wir beginnen also mit

```
octave> T = [ 0 0 0 ( 0:10 ) 10 10 10 ];X = Greville( 3,T );
```

und betrachten die Glättung einer zufällig gestörten linearen Funktion

```
octave> y = X .* ( 1 + .4*( rand( size(X) ) .- 1 ) );
```

Die Störung liefert einen relativen Fehler von höchstens 20%. Um für vorgegebenes $\lambda > 0$ und gleichmäßige[93] Gewichte das Minimierungsproblem zu lösen, berechnen wir zuerst die beiden Matrizen

```
octave> A = BSplVander( 3,T,X ); A = A*A';
octave> B = BSplGramDer( 3,T,2 );
```

und die rechte Seite

```
octave> yy = BSplVander( 3,T,X ) * y';
```

Wir plotten y

```
octave> clearplot; plot( X,y,"*" )
```

definieren eine Punktmenge zum Plotten des Splines,

```
octave> Z = (min(T):.01:max(T));
```

und berechnen und plotten den Glättungsspline durch Aufruf von

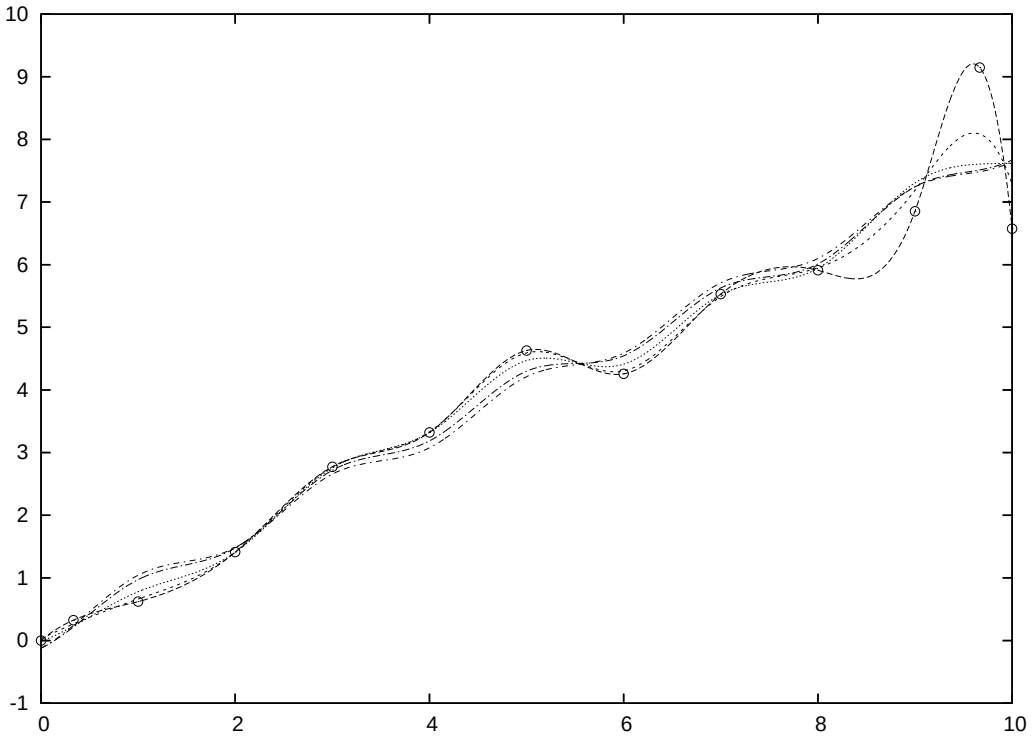


Abbildung 5.2: Der Glättungsspline für die Parameterwerte $\lambda = 0, 0.01, 0.1, 1, 10$.

```
octave> l = 0; d = ( ( A + l*B ) \ yy )';
octave> s = d*BSplVander( 3,T,Z ); plot( Z,s );
```

Der Effekt der Vergrößerung des Glättungsparameters kann sehr schön in Abb. 5.2 gesehen werden: Die Abweichung von den Daten wächst, aber die Glattheit des Splines, wieder im Sinne von reduzierten Oszillationen, wird im Gegenzug vergrößert. Übrigens hat dieser Prozess noch eine bemerkenswerte Eigenschaft:

> Die Lösung für $\lambda = 0$ ist der gute alte natürliche kubische Spline, die Lösung für $\lambda \to \infty$ hingegen die **lineare Regression**, d.h. die eindeutige Gerade, die minimalen Least-Squares-Abstand von den Daten hat.

[93] Also nicht vorhandene

5.1.7 Ein gleichmäßiger Glättungsspline

Wir können die Idee des Glättungsspline,

> Gewichte Approximationsgüte gegen Glätte

mit Minimierung der ∞-Norm kombinieren, was uns wieder zu einem linearen Programm[94] führen wird. Das heißt, wir betrachten jetzt das Problem

$$\min_{\mathbf{d}} \left(\max_{x \in X} \left\| S_m \mathbf{d}(x) - \mathbf{y}_x \right\| + \lambda \max_{j=1,\ldots,n-k} \left\| \left(\mathbf{d}\, \widehat{\mathbf{G}}_k \right)_j \right\| \right). \tag{5.12}$$

Wir wissen ja bereits aus (5.7), wie man aus einem **Minimax-Problem** ein lineares Optimierungsproblem macht und die Randbedingungen, die vom Glattheitsterm kommen, sind auch praktisch wie dort, abgesehen davon, daß wir den Interpolationsterm fallen lassen:

$$\begin{bmatrix} \widehat{\mathbf{G}}_k^T & -\mathbf{1}_{n-k \times d} \\ -\widehat{\mathbf{G}}_k^T & -\mathbf{1}_{n-k \times d} \end{bmatrix} \begin{bmatrix} \mathbf{d}^T \\ \nu \end{bmatrix} \leq \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \end{bmatrix} \tag{5.13}$$

Um den ersten Term, den Approximationsteil, zu behandeln, fahren wir genau wie oben fort, und setzen

$$u = \max_{x \in X} \left\| S_m \mathbf{d}(x) - \mathbf{y}_x \right\| = \max_{x \in X} \left\| \mathbf{d}\, \mathbf{N}_m(x) - \mathbf{y}_x \right\|,$$

also[95]

$$-u\mathbf{1} \leq \mathbf{d}\mathbf{N}_m(X) - \mathbf{y} \leq u\mathbf{1} \qquad \text{bzw.} \qquad \begin{array}{rcl} \mathbf{d}\mathbf{N}_m - u\mathbf{1} & \leq & \mathbf{y} \\ \mathbf{d}\mathbf{N}_m + u\mathbf{1} & \geq & \mathbf{y} \end{array}$$

was uns entsprechend

$$\begin{bmatrix} \mathbf{N}_m^T(X) & -\mathbf{1}_{\#X \times d} \\ -\mathbf{N}_m^T & -\mathbf{1}_{\#X \times d} \end{bmatrix} \begin{bmatrix} \mathbf{d}^T \\ u \end{bmatrix} \leq \begin{bmatrix} \mathbf{y}^T \\ -\mathbf{y}^T \end{bmatrix} \tag{5.14}$$

liefert. Unter Weglassen der Indizes an den $\mathbf{1}$-Matrizen[96], erhalten wir somit schließlich, daß der *Sup-Smoothing-Spline*[97], der (5.12) löst, als Lösung des linearen Programms

[94] Das ist inzwischen keine Überraschung mehr oder sollte zumindest keine mehr sein.

[95] Nur zur Erinnerung: $\mathbf{d}\mathbf{N}_m(X)$ und $\mathbf{y}$ sind beide X-Vektoren, also aus $\mathbb{R}^X \simeq \mathbb{R}^{\#X}$.

[96] Deren Dimension sollte aus dem Kontext heraus klar sein und die Indizes werden nun wirklich langsam nervig.

[97] Kurzschreibweise für „Smoothing Spline in der Supremumsnorm".

$$\min u + \lambda v, \qquad \begin{bmatrix} \mathbf{N}_m^T(X) & -\mathbf{1} & \mathbf{0} \\ -\mathbf{N}_m^T & -\mathbf{1} & \mathbf{0} \\ \widehat{\mathbf{G}}_k^T & \mathbf{0} & -\mathbf{1} \\ -\widehat{\mathbf{G}}_k^T & \mathbf{0} & -\mathbf{1} \end{bmatrix} \begin{bmatrix} \mathbf{d}^T \\ u \\ v \end{bmatrix} \leq \begin{bmatrix} \mathbf{y}^T \\ -\mathbf{y}^T \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix}. \tag{5.16}$$

Und dieses Optimierungsproblem können wir nun direkt in `lp_solve` stecken und so die Glättungssplines berechnen. Wir vergleichen das Verhalten der

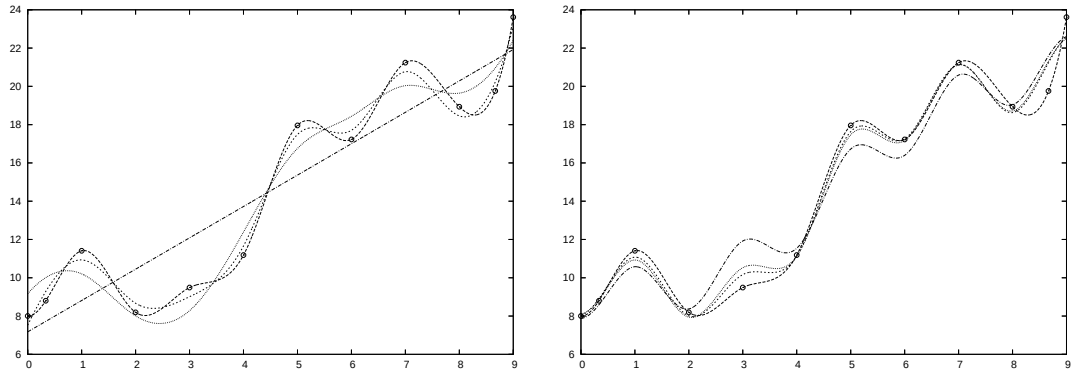


Abbildung 5.3: Glättungssplines für $p = \infty$ *(links)* und $p = 2$ *(rechts)* sowie die Parameterwerte $\lambda = 0, 0.05, 0.1, 1$. Diese Werte kann man ntürlich nicht direkt vergleichen, aber man sieht schon, daß der Glättungsspline bezüglich der ∞-Norm eher etwas mehr Wert auf Glattheit legt.

Glättungssplines für die beiden Normen und verschiedene Werte des Parameters λ in Abb. 5.3. Offensichtlich hat der Supremums-Spline eine wesentlich stärkere Tendenz, die Glattheit zu betonen, so man ihn den lässt, also wenn man den Glättungsparameter nicht ganz klein hält. Ein letztes Beispiel sehen wir in Abb. 5.4, diesmal wieder für die Betragsfunktion. Gerade die „mittleren" Werte von λ liefern gute und vor allem glättende Approximationen, die 2-Norm würde das nicht schaffen, da für sie eng lokalisierte Ausreisser und ansonsten ein gutes Verhalten ja besser sind als global kleine Werte.

5.2 Einfügen und Entfernen von Knoten

Der Satz von Curry und Schoenberg, Satz 3.11 sagt uns, daß ein Spline eine stückweise polynomiale Funktion von einer gewissen globalen Glattheit ist, die ihrerseits wieder von der Vielfachheit der Knoten abhängt. Trivialerweise ist

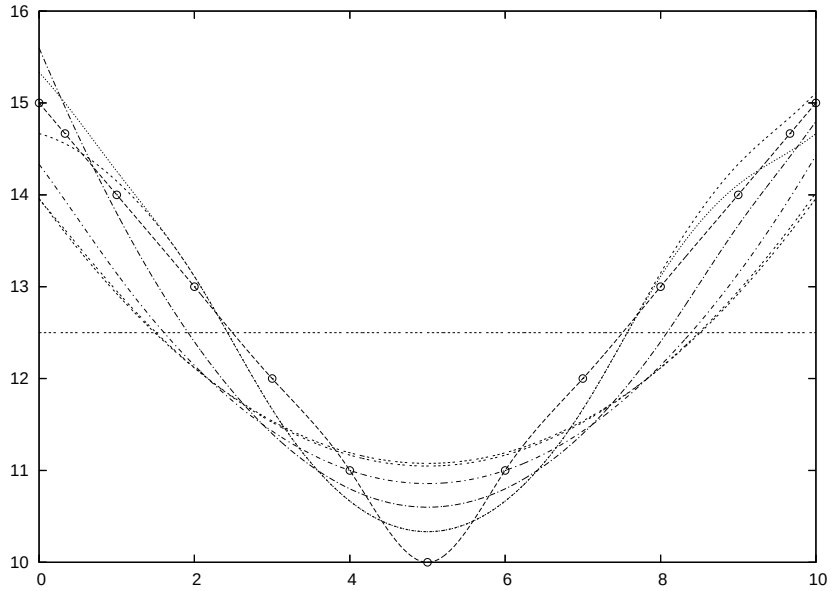


Abbildung 5.4: Approximation der Betragsfunktion für $p = \infty$ und $\lambda = 0, .5, 1, 2, 3, 4, 5, 10$.

aber ein Polynom natürlich auch ein stückweises Polynom - mit dem kleinen Zusatz, daß man an den entsprechenden Knoten eben einen C^m- oder C^∞-Übergang hat, was ja bei Polynomen vom Grad m dasselbe ist. Diese banale Bemerkung hat die folgende Konsquenz:

> Wenn man Knoten in eine Knotenfolge einfügt, dann ist jeder Spline bezüglich der groben Knotenfolge auch ein Spline bezüglich der verfeinerten Knotenfolge.

5.2.1 Verfeinerung von Knotenfolgen

Was aber bedeutet es, eine Knotenfolge zu verfeinern? Intuitiv müssen zwei Forderungen erfüllt werden:

1. Die verfeinerte Folge sollte jeden Knoten der Ausgangsfolge auch wieder enthalten.
2. Die Vielfachheit jedes Knoten in der verfeinerten Folge muss mindestens so groß sein wie in der originalen Knotenfolge.

Es gibt also zwei Methoden, eine Knotenfolge zu verfeinern:

Einfügen eines *neuen* Knotens: ersetze T durch T^* wobei

$$T^* = \{t_1, \ldots, t_j, t^*, t_{j+1}, \ldots, t_{m+n+1}\}, \qquad t_j < t^* < t_{j+1}.$$

Erhöhen der Vielfachheit: ersetze T durch T^* wobei

$$T^* = \{t_1, \ldots, t_j, t_j, t_{j+1}, \ldots, t_{m+n+1}\}, \qquad t_j < t_{j+1}.$$

Wir nennen $T^* = T^*_{m,n^*}$ eine **Verfeinerung** von T, geschrieben als $T \subseteq T^*$, wenn T^* ebenfalls eine gültige[98] Knotenfolge der Ordnung m ist und wenn es eine strikt monoton steigende Funktion $\nu : \{1, \ldots, n + m + 1\} \to \{1, \ldots, n^* + m + 1\}$ gibt, so daß $t^*_{\nu(j)} = t_j$, $j = 1, \ldots, n + m + 1$ ist. Diese Definition ist nur eine Formalisierung der oben erwähnten „intuitiven" Kriterien.

Es stellt sich die Frage, ob es sinnvoll ist, eine Knotenfolge „außerhalb" der Randknoten zu verfeinern bzw. zu erweitern. Die obige Definition schließt ein derartiges Vorgehen nicht aus, ermutigt aber auch nicht explizit dazu.

Jedes stückweise Polynom auf T ist offensichtlich auch ein stückweises Polynom auf $T^* \supseteq T$ und da die Vielfachheiten von Knoten beim Übergang von T zu T^* nur erhöht werden können, werden die Glattheitsforderungen an den Knoten *abgeschwächt*. Das führt unmittelbar zur folgenden Beobachtung:

Ist $T \subseteq T^*$, dann ist auch $\mathbb{S}_m(T) \subseteq \mathbb{S}_m(T^*)$.

Die B-Splines bezüglich T^* bilden eine Basis von $\mathbb{S}_m(T^*) \supseteq \mathbb{S}_m(T)$ und da jedes Element des Unterraums auch bezüglich der Basis des größeren Raums darstellbar ist, muss es zu jedem Kontrollpunktvektor $\mathbf{d}$ auch Kontrollpunkte $\mathbf{d}^*$ geben[99], so daß

$$\sum_{j=1}^{n} \mathbf{d}_j \, N_j^m(\cdot \,|\, T) = \sum_{j=1}^{n^*} \mathbf{d}^*_j \, N_j^m(\cdot \,|\, T^*),$$

und die offensichtliche Frage besteht natürlich darin, wie man $\mathbf{d}^*$ berechnen kann - und zwar am besten effektiv.

[98] Das heißt, daß auch nach dem Einfügen die Vielfachheit keines Knotens $m+1$ überschreiten darf.

[99] Die Umkehrung gilt natürlich nicht!

5.2.2 Knoteneinfügen

Wir werden uns hier auf eine Methode beschränken, die *einzelne Knoten* einfügt, normalerweise als **Boehm-Algorithmus**[100] bezeichnet und im Gegensatz zum sogenannten **Oslo-Algorithmus**, der mehrere Knoten simultan einfügen kann und auf *diskreten B-Splines*[101] basiert.
Wir betrachten die folgende Situation:

> Füge einen Knoten t^* zwischen zwei verschiedenen Knoten $t_j < t_{j+1}$ ein, wobei $t_j \le t^* < t_{j+1}$ ist. Wenn wir also die Vielfachheit eines Knotens erhöhen, dann tun wir das „von rechts".

Die erste Beobachtung ist, daß Knoteneinfügen ein *lokaler* Prozess ist und auch sein muss, denn Knoteneinfügen betrifft ja nur diejenigen B-Splines, deren Träger das Intervall $[t_j, t_{j+1}]$ enthält und das sind genau die B-Splines $N^m_{j-m}, \ldots, N^m_j$. Das bedeutet aber auch, daß nur die zugehörigen Kontrollpunkte $\mathbf{d}_{j-m}, \ldots, \mathbf{d}_j$ für das Knoteneinfügen relevant sein werden. Da außerdem das Einfügen eines *einzelnen* Knotens $n^* = n + 1$ liefert, enthält $\mathbf{d}^*$ genau einen Koeffizienten mehr als $\mathbf{d}$.

Algorithmus 5.1 (Knoteneinfügen)
Eingabe:

- *Knotenfolge* $T = T_{m,n}$.
- *Einzufügender Knoten* $t_j \le t^* < t_{j+1}$.

Prozedur:

1. *Für* $k = 1, \ldots, j - m$ *setze*
$$\mathbf{d}^*_k = \mathbf{d}_k.$$
2. *Für* $k = j - m + 1, \ldots, j$
   (a) *Berechne*
   $$\alpha_k = \frac{t_{k+m} - t^*}{t_{k+m} - t_k}.$$
   (b) *Setze*
   $$\mathbf{d}^*_k = \alpha_k \, \mathbf{d}_{k-1} + (1 - \alpha_k) \, \mathbf{d}_k.$$

[100] Eigentlich hieß Herr Boehm, Geometer in Braunschweig, Böhm, zumindest bis er beschloss, daß ein Umlaut einer internationalen Karriere nicht foerderlich wäre.
[101] Die sind nun wieder etwas mathematisch sehr ansprechendes und reizvolles, aber werden leider dennoch nicht in dieser Vorlesung auftauchen.

3. *Für* $k = j+1, \ldots, n+1$ *setze*

$$\mathbf{d}^*_k = \mathbf{d}_{k-1}.$$

Ergebnis: *Kontrollpunkte* $\mathbf{d}^*$ *so daß*

$$S_m\mathbf{d}^*(\cdot \,|\, T^*) = S_m\mathbf{d}(\cdot \,|\, T).$$

Wieder einmal können wir lineare Algebra benutzen, um das Vorgehen, also die Regel

$$\mathbf{d}^*_k = \frac{t_{k+m} - t^*}{t_{k+m} - t_k}\,\mathbf{d}_{k-1} + \frac{t^* - t_k}{t_{k+m} - t_k}\,\mathbf{d}_k, \qquad k = j-m+1, \ldots, j, \tag{5.17}$$

zu formalisieren, und zwar als die Matrixmultiplikation

$$\mathbf{d}^* = \mathbf{d}\mathbf{A}(t^*), \qquad \mathbf{A}(t^*) = \begin{bmatrix} 1 & & & & & & & & & \\ & \ddots & & & & & & & & \\ & & 1 & \alpha_{j-m+1} & & & & & & \\ & & & 1-\alpha_{j-m+1} & \ddots & & & & & \\ & & & & \ddots & \alpha_j & & & & \\ & & & & & 1-\alpha_j & 1 & & & \\ & & & & & & & \ddots & & \\ & & & & & & & & 1 & \end{bmatrix} \in \mathbb{R}^{n\times n+1}.$$

Beachten Sie, daß $\mathbf{A}(t^*)$ eine sehr dünn besetzte bandierte Matrix ist. Die Funktion, die diese Matrix bestimmt heißt `KinsMat`, das Paar $\mathbf{A}, T^*$ liefert die Funktion `KInsert`. Schauen wir uns doch eines unserer früheren Beispiele nochmal an, und zwar

```
octave> T = [ 0,0,0,0,1,2,3,4,4,4,4 ];
octave> d = [ 0 1 1 .5 0 0 1 ; 0 .4 .8 1 .8 .4 0 ];
```

und fügen den Knoten $t^* = 2.5$ ein. Das neue Kontrollpolygon bestimmt sich dann einfach als

```
octave> [A,Tt] = KInsert( 3,T,2.5 ); dd = d * A;
```

Das Ergebnis eines einfachen Knoteneinfügens ist in Abb. 5.5 zu sehen. Da Knoteneinfügen immer eine **Konvexkombination** der ursprünglichen Kontrollpunkte bildet, liegt das neue Kontrollpolygon innerhalb deren konvexer Hülle.

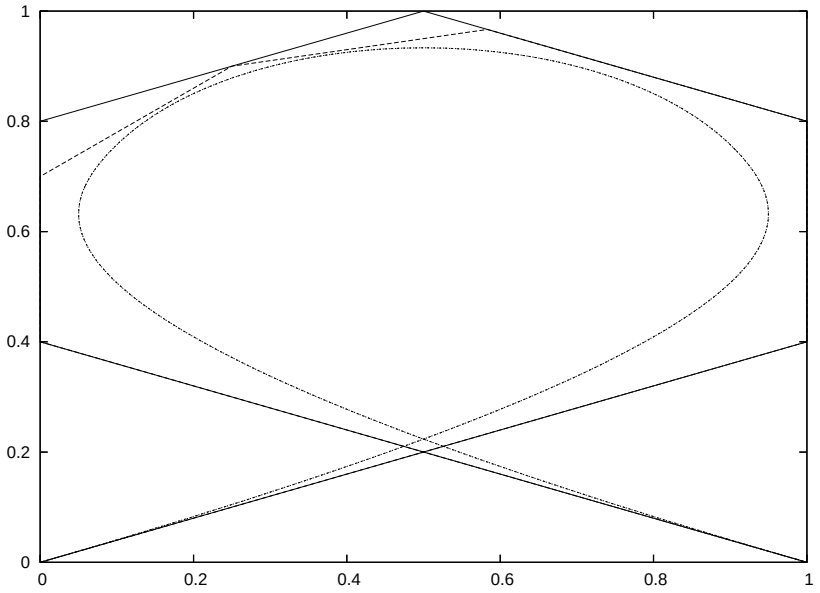


Abbildung 5.5: Knoteneinfügen. Die beiden Kontrollpolygone und die Splines sind dargestellt, aber natürlich sieht man nur eine Splinekurve, denn die beiden Kurven sind ja identisch - schließich ist es die Grundidee des Knoteneinfügens, daß sich die Kurve eben **nicht** ändert.

Knoteneinfügen kann man wiederholen[102]. Beispielsweise können wir einen Knoten m Mal einfügen und da Splinefunktionen an m-fachen Knoten immer *interpolieren*[103], gibt uns das noch ein Verfahren, einen Spline an dem neueingefügten Knoten *auszuwerten* - das sich allerdings bei genauem Hinsehen auch wieder als Algorithmus von de Boor entpuppt. Im numerischen Experiment:

```
octave> d = [ 0 1 1 .8 .2 0 0 1 ; 0 .2 .6 1 1 .6 .2 0 ];
octave> T = [ 0,0,0,0,1,2,3,4,5,5,5,5 ];
octave> for j=1:3 [ A,T ] = KInsert( 3,T,2.5 ); d = d*A; end
```

und jetzt liegt einer der Kontrollpunkte auf der Kurve, wie man in Abb. 5.6 sehen kann.

[102]Nicht für die Prüfung oder zumindest nicht unbedingt, sondern im Sinne von iterierter Ausführung.

[103]Wir haben bisher diese Eigenschaft gar nicht erwähnt, die recht unmittelbar aus dem Algorithmus von de Boor folgt, da ein Knoten der Vielfachheit m immer am linken Rand aller betrachteten Referenzintervalle (für die baryzentrischen Koordinaten) liegt, so daß der Kontrollpunkt $\mathbf{d}_{j-m}$ in allen Schritten des Algorithmus reproduziert wird.

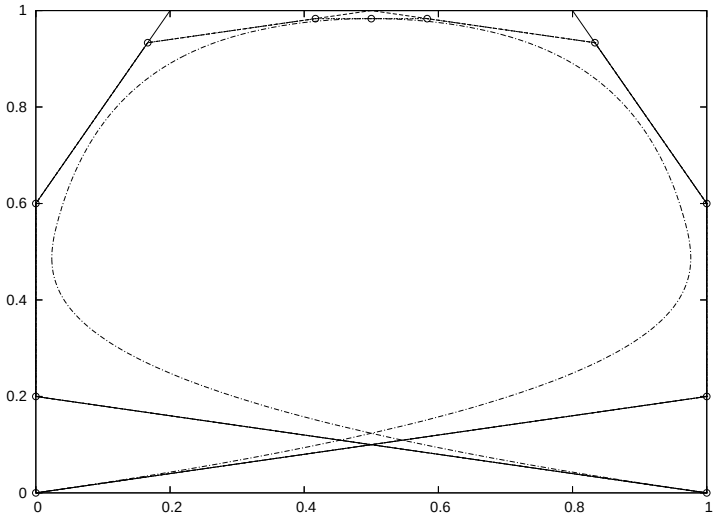


Abbildung 5.6: Ein Beispiel dreifachen Knoteneinfügens mit den „Zwischenpunkten“ und den „endgültigen“ Kontrollpunkten. Einer von diesen liegt auch tatsächlich auf der Kurve und zwar genau dort, wo der dreifache Knoten eingefügt wurde.

Unser Ansatz vermittels linearer Algebra hat einen netten Nebeneffekt, der es uns erlaubt, **virtuelles Knoteneinfügen** durchzuführen: Anstatt das Knotrollpolygon und die Knotenfolge zu verändern, speichern wir lediglich die (bandierte) Matrix

$$\mathbf{A}\left(t_1^*, \ldots, t_k^*\right) = \mathbf{A}\left(t_k^*\right) \cdots \mathbf{A}\left(t_1^*\right)$$

die das Knoteneinfügen beschreibt; allerdings sollte man nicht übersehen, daß $\mathbf{A}\left(t_j^*\right)$ von der Knotenfolge $T \cup \left\{t_1^*, \ldots, t_{j-1}^*\right\}$ abhängt, so daß es keine Kommutativitätsbeziehungen unter den Knoteneinfügematrizen gibt.

5.2.3 Erste Anwendungen des Knoteneinfügens

Knoteneinfügen hat viel mehr zu bieten als „nur“ eine Erhöhung der Flexibilität einer Splinekurve durch Hinzufügen weiterer Kontrollpunkte, obwohl das natürlich erst einmal das Hauptziel bei der ursprünglichen Entwicklung dieser Verfahren war.

Als erste Anwendung betrachten wir die Konvertierung einer Splinekurve in ihre **stückweise polynomiale Form** oft auch als **PP-Form**[104] der Kurve

[104] Für *piecewise polynomial* ...

bezeichnet. Die Polynomstücke könnte man nur beispielsweise bezüglich der Monombasis darstellen, aber da diese ja keine allzu große geometrische Relevanz hat, können wir auch genauso gut[105] auf die **Bézier–Darstellung** einer polynomialen Kurve auf dem nichtdegenerierten Intervall $[t_j, t_{j+1}]$ verweisen:

$$\mathbf{p}(x) = B_n\mathbf{d}(x) = \sum_{j=0}^{n} \mathbf{d}_j \binom{n}{j} \lambda^j(x)\,(1-\lambda(x))^{n-j}\,, \qquad \lambda(x) = \frac{x - t_j}{t_{j+1} - t_j}.$$

Die Bézier-Darstellung eines Polynoms auf $[a, b]$ entspricht dem einfachsten Typ einer Splinefunktion, nämlich der mit $m+1$-fachen Knoten an a und b und nirgendwo sonst. Um also die Restriktion einer Splinekurve auf das Intervall $[a, b]$, $t_j \le a < b \le t_{j+1}$, zu bestimmen, haben wir nun ein ganz einfaches Rezept:

> Die Einschränkung einer Splinekurve auf ein Intervall, dessen inneres keinen Knoten enthält, kann dadurch bestimmt werden, daß man beide Endpunkte des Intervalls solange in die Knotenfolge einfügt, bis sie Vielfachheit $m+1$ haben.

Zwar gibt Knoteneinfügen „nur“ die Koeffizienten der Bézier-Darstellung, aber die kann man nun, wenn nötig, relativ einfach in eine **Monomdarstellung** konvertieren. Die PP-Darstellung hat durchaus einige Vorteile:

- NC-Maschinen zum Fräsen oder Laserschneiden[106] „verstehen“ stückweise Polynome, zumindest bis zum Grad 5, so daß man die Polynomkoeffizienten direkt in so eine NC–Maschine einspielen kann.
- Der lokale Auswertungsalgorithmus kann nun mit einer Komplexität von $O(m)$ durchgeführt werden[107] während der Algorithmus von der Boor immer noch bei $O(m^2)$ liegt.Wenn also ein Spline sehr oft an sehr vielen Stellen auszuwerten ist, dann kann so eine Konvertierung durchaus den Aufwand wert sein.

Die Spline-Toolbox[108] in `Matlab` hat im Übrigen integrierte Umwandlungsroutinen zwischen der B-Spline- und PP-Darstellung von Splines, siehe (Boor, 1978).

Die nächste Anwendung ist der **Vergleich von Splinekurven** wobei wir die beiden Splinekurven

$$S_m\mathbf{d}\,(\cdot\,|\,T) = \sum_{j=1}^{n} \mathbf{d}_j\, N_j^m\,(\cdot\,|\,T) \qquad \text{und} \qquad S_m\mathbf{d}'\,(\cdot\,|\,T') = \sum_{j=1}^{n'} \mathbf{d}'_j\, N_j^m\,(\cdot\,|\,T')$$

[105] Wenn nicht noch besser!
[106] Genauer: Die zugehörige `840D`-Steuerung der Siemens AG.
[107] Stichwort: Hornerschema!
[108] Von C. de Boor programmiert.

betrachten und miteinander vergleichen wollen. Sei $T^* = T \cup T'$ die Vereinigung der beiden Knotenfolgen, das heißt, die *kleinste* Knotenfolge mit der Eigenschaft $T \subseteq T^*$ und $T' \subset T^*$, dann können wird die jeweils „fehlenden" Knoten in die beiden Folgen einfügen und sie als

$$S_m\mathbf{d}\,(\cdot\,|\,T) = S_m\mathbf{d}\mathbf{A}\,(T^* \setminus T)\,(\cdot\,|\,T^*)$$

sowie

$$S_m\mathbf{d}'\,(\cdot\,|\,T') = S_m\mathbf{d}'\mathbf{A}\,(T^* \setminus T')\,(\cdot\,|\,T^*)$$

schreiben, womit sich ihre Differenz nun als

$$\left\|\mathbf{d}\mathbf{A}\,(T^* \setminus T) - \mathbf{d}'\mathbf{A}\,(T^* \setminus T')\right\|$$

abschätzen lässt; als Matrixnorm können wir hier entweder die **Frobeniusnorm**

$$\begin{aligned}&\left\|\mathbf{d}\mathbf{A}\,(T^* \setminus T) - \mathbf{d}'\mathbf{A}\,(T^* \setminus T')\right\|_F \\ &= \left(\sum_{j=1}^{d}\sum_{k=1}^{n^*}\left|(\mathbf{d}\mathbf{A}\,(T^* \setminus T))_{j,k} - (\mathbf{d}'\mathbf{A}\,(T^* \setminus T'))_{j,k}\right|^2\right)^{1/2}\end{aligned} \tag{5.18}$$

oder die **Vektor-Supremumsnorm**

$$\begin{aligned}&\left\|\mathbf{d}\mathbf{A}\,(T^* \setminus T) - \mathbf{d}'\mathbf{A}\,(T^* \setminus T')\right\|_\infty \\ &= \max_{j=1,\ldots,d}\max_{k=1,\ldots,n^*}\left|(\mathbf{d}\mathbf{A}\,(T^* \setminus T))_{j,k} - (\mathbf{d}'\mathbf{A}\,(T^* \setminus T'))_{j,k}\right|.\end{aligned} \tag{5.19}$$

verwenden. Damit können wir einen Spline $S_m\mathbf{d}'\,(\cdot\,|\,T')$ durch einen anderen Spline $S_m\mathbf{d}\,(\cdot\,|\,T)$ mit einer möglicherweise komplett anderen Knotenfolge[109] approximieren, indem wir wieder einmal eines unserer Glättungsprobleme lösen, also beispielsweise

$$\min_{\mathbf{d}}\left\|\mathbf{d}\mathbf{A}\,(T^* \setminus T) - \mathbf{d}'\mathbf{A}\,(T^* \setminus T')\right\|_F^2 + \sum_{r=1}^{m-1}\lambda_r\int\left\|S_m^{(r)}\mathbf{d}\right\|^2 \tag{5.20}$$

oder

$$\min_{\mathbf{d}}\left\|\mathbf{d}\mathbf{A}\,(T^* \setminus T) - \mathbf{d}'\mathbf{A}\,(T^* \setminus T')\right\|_\infty + \sum_{r=1}^{m-1}\lambda_r\int\left\|\mathbf{d}\widehat{\mathbf{G}}_r\right\|_\infty, \tag{5.21}$$

und zwar mit den Glättungsspline-Methoden aus den vorhergenden Kapiteln. Für hinreichend kleine Werte von λ werden sich diese Approximanten dem

[109] Beispielsweise könnten die Knoten in T enstprechend der Krümmung des Splines $S_m\mathbf{d}'$ verteilt sein, siehe (Boor, 1978).

Ausgangsspline nähern während für größere Werte von λ das Gewicht auf Oszillationsverminderung gelegt werden wird.

Es ist wichtig, zu betonen, daß die Minimierung nur bezüglich **d** durchgeführt wird - das Knoteneinfügen wird komplett von der Matrix $\mathbf{A}(T^* \setminus T)$ behandelt, ein weiteres schönes Beispiel für **virtuelles Knotenneinfügen**.

5.2.4 Nullstellen von Splinefunktionen

Eine etwas unerwartete Anwendung des Knoteneinfügens ist die Bestimmung von Nullstellen bzw., etwas allgemeiner und im Kurvenfall, die Aufgabe, den **Schnittpunkt** einer Splinekurve mit einer gegebenen Gerade zu ermitteln. Auch letzteres ist im Wesentlichen „nur" das Problem, die Nullstelle einer Splinekurve zu finden: Sind $\mathbf{a}, \mathbf{b} \in \mathbb{R}^d$ Anfangs- und Endpunkte der Geraden, dann setzen wir

$$\mathbf{l}_j = \frac{t_{n+1} - x_j}{t_{n+1} - t_{m+1}}\,\mathbf{a} + \frac{x_j - t_{m+1}}{t_{n+1} - t_{m+1}}\,\mathbf{b}, \qquad j = 1, \ldots, n,$$

wobei die x_j wieder einmal die **Greville-Abszissen** zur Knotenfolge T sind, dann ist $S_m\mathbf{l}$ eine lineare Funktion[110] und damit hat der Spline $S_m\mathbf{d} - S_m\mathbf{l} = S_m(\mathbf{d} - \mathbf{l})$ genau dort eine Nullstelle wo $S_m\mathbf{d}$ die Linie schneidet.

Die Standardmethode zur Berechnung von **Nullstellen** einer Funktion ist das **Newton-Verfahren**, das eine Nullstelle mittels der Iteration

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}, \qquad k = 0, 1, 2, \ldots,$$

und passendem[111] Anfangswert x_0 annähert. Allerdings hat das Newtonverfahren ein paar wohlbekannte Probleme:

1. Der Erfolg der Iteration hängt sehr stark von der Qualität des Startwerts x_0 ab, denn bekanntlich konvergiert das Verfahren nur lokal. Einen guten Startwert zu finden ist oftmals nicht leicht.

2. Für die Durchführung der Iteration muß in jedem Schritt die *Ableitung* des Splines mitberechnet werden. Das ist sogar noch unser kleinstes Problem, denn zumindest der Algorithmus von de Boor kann relativ einfach so erweitert werden, daß er diesen Wert mitliefert.

[110] Genaugenommen ist es der Schoenbergoperator angewandt auf die lineare Funktion, die **a** und **b** verbindet.

[111] Das ist so einfach dahingesagt ...

3. Wenn wir mit Spline*kurven* arbeiten, dann haben wir es mit *vektorwertigen* Funktionen zu tun und die obige Iteration ist noch nicht einmal wohldefiniert[112]. Die naheliegendste Idee wäre mit Sicherheit eine *komponentenweise* Behandlung der Kurve, aber man sollte sich dann schon vor Augen halten,daß wir dann eine **simultane Nullstelle** aller Komponenten bestimmen müssen und das vieles, was in einer Komponente eine Nullstelle liefert, durch die anderen Komponenten für nutzlos erklärt werden kann.

Diese Schwierigkeiten kann man umgehen, wenn man einen Algorithmus verwendet, der auf Knoteneinfügen basiert und auf Mørken und Reimers (Mørken & Reimers,) zurückgeht. Wir werden hier nur die grundlegende Idee beschreiben, Details und ein Konvergenzbeweis[113] sind in (Mørken & Reimers,) zu finden. Die Idee hinter dieser Methode ist in der Tat sehr intuitiv:

> Schneide das *Kontrollpolygon* mit der Linie und füge die zu dem Schnittpunkt gehörige Abszisse als neuen Knoten in die Knotenfolge ein.

Gibt es **keinen** Schnittpunkt zwischen Kontrollpolygon und Linie, dann gibt es auch keinen Schnittpunkt zwischen der Kurve und der Linie[114], weil die Kurve ja in der konvexen Hülle des Kontrollpolygons verlaufen muss. Generell sagt die **Variationsverminderung** durch Splines, siehe (Marsden & Schoenberg, 1966; Marsden & Schoenberg, 1988), daß jede Hypereben im $\mathbb{R}^d$ mindestens so viele[115] Schnittpunkte mit dem Kontrollpolygon wie mit der Kurve haben muss. Wenn es also keinen Schnitt mit dem Kontrollpolygon gibt, dann gibt es auch keinen Schnitt mit der Kurve und wir können aufhören, nach so etwas suchen zu wollen.

Wenn andererseits das Kontrollpolygon die Gerade schneidet oder irgendwo eine Nullstelle hat[116], dann passiert das zwischen zwei Kontrollpunkten[117],

[112] Wie dividiert man durch einen Vektor. Bei einer quadratischen Matrix hat man ja noch eine Idee, aber ein Vektor?

[113] Und zwar ein Beweis für *quadratische Konvergenz*, die Methode ist also mit dem Newtonverfahren vergleichbar, was die Geschwindigkeit angeht.

[114] Für $d > 2$ muss man ein wenig vorsichtiger sein, es sind Dinge wie winschiefe Geraden, die uns das Leben dann etwas schwerer machen.

[115] Im „Normalfall" wohl sogar mehr!

[116] Eben je nachdem ob wir Spline**kurven** oder Spline**funktionen** betrachten.

[117] Der Fall, daß der Kontrollpunkt selbst der Schnittpunkt ist, ist etwas singulär und wird normalerweise keinen Schnittpunkt ergeben, außer im Falle m-facher Knoten oder „entarteter" Kontollpolygone. Mit anderen Worten: Den Sonderfall muss man als solchen und mit entsprechender Sorgfalt behandeln.

sagen wir $\mathbf{d}_j$ und $\mathbf{d}_{j+1}$, genauer, an der Stelle

$$\lambda\,\mathbf{d}_j + (1-\lambda)\;\mathbf{d}_{j+1}, \qquad \lambda \in [0,1].$$

Motiviert durch den Schoenbergoperator fassen wir den Spline als $S_m\mathbf{d} = \mathscr{S}_m\mathbf{f}$ für eine stetige Funktion $\mathbf{f}$ mit der Eigenschaft $\mathbf{d}_k = \mathbf{f}(x_k)$ auf und erhalten so eine natürliche Beziehung zwischen den Kontrollpunkten und den Greville-Abszissen. Daher ist eine gute[118] Näherung für die Nullstelle durch den Punkt

$$t^* = \lambda\,x_j + (1-\lambda)\;x_{j+1},$$

gegeben, den wir dann als neuen oder weiteren Knoten in T einfügen. Das liefert ein verfeinertes Kontrollpolygon und der ganze Prozess wird wiederholt, bis man ein Kontrollpolygonstück hat, das nahe genug bei der Geraden oder eben bei Null liegt. Betrachtet man in jedem Iterationsschritt übrigens *alle* Schnittpunkte des Kontrollpolygons, dann findet man so letztendlich auch *alle* Schnittpunkte der Splinefunktion.

Eine wichtige Anwendung der Nullstellenbestimmung findet statt, wenn ein **nächster Punkt** zu bestimmen ist: Zu $\mathbf{y} \in \mathbb{R}^d$ und einer Splinekurve $S_m\mathbf{d}$, definiert, wie immer, durch Knotenfolge und Kontrollpolygon, soll man den nächsten Punkt auf der Kurve bezüglich der euklidischen Norm finden, also das Minimierungsproblem

$$\min_x \left\| S_m\mathbf{d}(x) - \mathbf{y} \right\|_2$$

lösen. Natürlich quadriert man hier mal als erstes die Zielfunktion und kann dann genausogut die Funktion

$$\begin{aligned}
\left\| S_m\mathbf{d}(x) - \mathbf{y} \right\|_2^2 &= (\mathbf{d}\,\mathbf{N}_m(x))^T\,(\mathbf{d}\,\mathbf{N}_m(x)) - 2\mathbf{y}^T\,(\mathbf{d}\,\mathbf{N}_m(x)) + \mathbf{y}^T\mathbf{y} \\
&= \sum_{j=1}^{d}\sum_{k,\ell=1}^{n} \mathbf{d}_{jk}\mathbf{d}_{j\ell}N_k^m(x)N_\ell^m(x) - 2\sum_{j=1}^{d}\sum_{k=1}^{n} \mathbf{d}_{jk}y_jN_k^m(x) + \sum_{j=1}^{d} y_j^2 \\
&= \sum_{k,\ell=1}^{n} \left(\mathbf{d}^T\mathbf{d}\right)_{k,\ell} N_k^m(x)N_\ell^m(x) - 2\sum_{k=1}^{n} \left(\mathbf{d}^T\mathbf{y}\right)_k N_k^m(x) + \left\|\mathbf{y}\right\|_2^2
\end{aligned}$$

minimieren. Die obige Funktion ist eine skalarwertige **Splinefunktion** vom Grad $2m$ mit Bruchstellen an den Knoten von T, sie gehört also zu $\mathbb{S}_{2m}(T^*)$, wobei

$$T^* = \{t_1, \ldots, T, t_{n+m+1}, \ldots, t_{n+m+1}\},$$

[118] Oder, korrekter, eine *linearisierte*. Und Newton linearisiert ja auch!

so daß nur die Koeffizienten der Randpunkte angepasst werden müssen. Also ist

$$\left\| S_m \mathbf{d}(x) - \mathbf{y} \right\|_2^2 = \sum_{j=1}^{n} c_j \, N_j^{2m}(x \mid T^*) = \mathbf{c}\, \mathbf{N}_{2m}(x \mid T^*) \,,$$

und um $\mathbf{c}$ zu berechnen benötigen wir zwei Operationen:

1. eine **Multiplikationsformel** für Splines,
2. eine **Graderhöhungsformel** für Splines,

aber an dieser Stelle begnügen wir uns mit der Feststellung, daß man beide Operationen tatsächlich ausführen kann, und zwar schnell. Um dann die Abszisse des gesuchten nächsten Punktes auf der Kurve zu finden, brauchen wir nurnoch die Nullstellesuche auf die Funktion

$$\frac{d}{dx} \left\| S_m \mathbf{d}(x) - \mathbf{y} \right\|_2^2 = \mathbf{c}\widehat{\mathbf{G}}_1 \, \mathbf{N}_{2m-1}\left(x \mid \widehat{T^*}\right)$$

anwenden.

5.2.5 Knotenentfernen

Knotenentfernen ist der inverse Prozess zum Knoteneinfügen - anstatt von T zu T^* überzugehen, möchten wir von T^* zu T „zurück". Nachdem immer noch $\mathbb{S}_m(T) \subset \mathbb{S}_m(T^*)$ gilt, können wir natürlich nicht erwarten, daß Knotenentfernen im allgemeinen exakt ausgeführt werden kann[119], so daß die resultierende Splinekurve die Ausgangskurve nur noch approximieren, nicht aber reproduzieren wird. Und diese Approximation versucht man eben so gut wie möglich zu machen.

Wir fangen jetzt also mit $S_m \mathbf{d}^*$ an und wollen das zugehörige $\mathbf{d}$ nach Entfernung von t^* aus T^* bestimmen. Um das zu tun, fügen wir t^* einfach wieder in die Knotenfolge ein und vergleichen das resultierende Kontrollpolygon $\mathbf{dA}(t^*)$ mit $\mathbf{d}^*$, wobei wir beispielsweise den Ausdruck

$$\left\| \mathbf{dA}(t^*) - \mathbf{d}^* \right\|_2^2 = \mathbf{dAA}^T\mathbf{d}^T - 2\mathbf{d}^*\mathbf{A}^T\mathbf{d}^T + \mathbf{d}^*(\mathbf{d}^*)^T$$

minieren, der wieder mal zu **Normalengleichungen** führt, und zwar zu

$$\mathbf{dAA}^T = \mathbf{d}^*\mathbf{A}^T.$$

[119] Außer natürlich wenn der Knoten vorher gerade erst eingefügt worden ist.

Das ist allerdings nicht der einzige Weg, das Kontrollpolygon eines „knotenentfernten“ Splines zu berechnen. Spätens seitdem wir einiges über Glättungssplines gelernt haben, können wir uns sofort eine ganze Menge von Erweiterungen und Verallgemeinerungen denken:

1. Verwendung einer anderen Norm, insbesondere $\left\|\mathbf{dA}\left(t^*\right) - \mathbf{d}^*\right\|_\infty$. Das wird natürlich wieder das eine oder andere nette lineare Optimierungsproblem liefern.

2. Minimierung bezüglich einer gewichteten Norm, die auch noch einen **Glattheitsterm** für $S_m\mathbf{d}$ enthält. Immerhin kann man ja das Knotenentfernen als Übergang von einer „komplizierten“ zu einer „einfacheren“ Kurve ansehen und es kann durchaus sinnvoll sein, wenn diese Kurve so glatt ist wie möglich.

3. Simulatenes Entfernen meherer Knoten - da ist dann nur die Matrix **A** ein bisschen komplizierter.

Kenntnis der Mittel ohne eine eigentliche Anwendung, ja ohne Gabe und Willen, sie anzuwenden, ist, was man jetzt gemeiniglich Gelehrsamkeit nennt

Lichtenberg, Philosophische Bemerkungen

6 Tensorproduktflächen

Bevor wir uns an die „richtigen“ und dann auch richtig multivariaten[120] Splines machen, wollen wir uns zuerst noch eine recht einfache aber weitverbreitete Methode ansehen, aus Kurven Flächen oder sogar höherdimensionale Objekte zu generieren.

6.1 Flächen als Kurven entlang Kurven

Genauso wie wir eine *paramterische* Kurve als Transformation des Parameterintervalls gesehen haben könnten wir eine **parametrische Fläche** als Transformation eines zweidimensionalen Paramtergebiets ansehen, also als Abbildung $\mathbf{f} : \Omega \to \mathbb{R}^d$, $\Omega \subset \mathbb{R}^2$. Das wird aber bereits in zwei Variablen kompliziert, denn während in einer Frage das Intervall eigentlich ohne weitere Fragen als kanonischer Parameterbereich durchging, haben wir jetzt die freie Auswahl: Kreise, Rechtecke, Dreiecke oder auch noch wesentlich komplexere Gebilde.

Ein intuitiv sehr schöner Ansatz zur Beschreibung von Flächen findet sich in P. Béziers Einleitungskapitel in (Farin, 1988):

> Eine Fläche entsteht dadurch, daß sich eine Kurve durch den Raum bewegt und dabei verändert.

[120] Es gibt durchaus unterschiedliche Meinungen in der „Fachwelt“, was die wirklich richtigen multivariaten Splines sind - das ist ein typisches Phänomen bei der Verallgemeinerung univariater Objekte, daß sich das sehr unterschiedlich entwickeln kann, je nachdem, welchen Aspekt man genau verallgemeinert.

Mathematisch entspricht dieses Flächenkonzept einer *einparametrischen Familie von Kurven* $\mathbf{f}_y : I \to \mathbb{R}^d$, wobei wir selbstverständlich für alle Werte von y *dasselbe* Intervall I wählen können und werden. Außerdem spricht nichts dagegen, für alle Kurven dieselbe mathematische Klasse zu verwenden, also beispielsweise Splines[121] der Ordnung m zu einer Knotenfolge T - und auch m und T sind wieder *unabhängig* von y, das sich dann nur noch auf die die Kontrollpunkte auswirken kann. Also:

$$\mathbf{f}_y(x) = \sum_{j=1}^{n} \mathbf{d}_{y,j}\, N_j^m(x \mid T) = \sum_{j=1}^{n} \mathbf{d}_j(y)\, N_j^m(x \mid T)\,, \tag{6.1}$$

wobei der Unterschied zwischen den beiden Schreibweisen ein rein formaler ist - wo wir das y hinschreiben, das ist ja eigentlich egal. Allerdings könnte uns bei der zweiten Darstellung ja eigentlich die Idee kommen, daß wir die Veränderung der Kontrollpunkte auch wieder als Kurve ansehen und daher $\mathbf{d}(y)$ auch wieder als Splinekurve der Ordnung m' mit Knotenfolge T' in der Form

$$\mathbf{d}_j(y) = \sum_{k=1}^{n'} \mathbf{d}_{jk}\, N_k^{m'}(y \mid T') \tag{6.2}$$

schreiben können. Nun setzen wir nur noch (6.2) in (6.1) ein, ersetzen n, m, T durch n_1, m_1, T_1 und entsprechend n', m', T' durch n_2, m_2, T_2 – dann wird der Ausdruck symmetrischer – und erhalten

$$\mathbf{f}(x, y) = \sum_{j=1}^{n_1} \sum_{k=1}^{n_2} \mathbf{d}_{jk}\, N_j^{m_1}(x \mid T_1)\; N_k^{m_2}(y \mid T_2)\,. \tag{6.3}$$

So, das wird uns nun aber entschieden zu indexlastig, und deswegen führen wir ein paar neue Schreibweisen ein.

Definition 6.1 (Tensorprodukt)

1. *Mit* $\mu = (m_1, m_2) \in \mathbb{N}_0^2$ *bezeichnen wir den* ***Multigrad*** *des Splines, mit* $\nu = (n_1, n_2) \in \mathbb{N}^2$ *die Anzahl der Kontrollpunkte in* x- *bzw.* y-*Richtung. Anstelle des Parameterwertes* (x, y) *schreiben wir* $x = (x_1, x_2)$.

2. *Für zwei Multiindizes* $\alpha, \beta \in \mathbb{N}^s$ *schreiben wir* $\alpha \le \beta$ *falls* $\alpha_j \le \beta_j$, $j = 1, \ldots, s$. *Diese Ordnung ist nur eine* ***Halbordnung***[122]

[121] Die Wahl fällt uns schon deswegen leicht, weil wir ja ohnehin nicht sonderlich viele Kurventypen kennen.

[122] Das heißt, es gibt Elemente wie beispielsweise $\alpha = (1, 0)$, $\beta = (0, 1)$, die unvergleichbar sind, das heißt, für die weder $\alpha \le \beta$ noch $\beta \le \alpha$ gilt.

3. *Das* ***Kreuzprodukt*** *der Knoten ist definiert als*

$$T := T_1 \otimes T_2 = \{t_\alpha = (t_{\alpha_1}, t_{\alpha_2}) \ : \ \alpha \le \nu + \mu + \mathbf{1}\}, \qquad \mathbf{1} = (1, \dots, 1), \quad (6.4)$$

das der ***Tensorprodukt*** *Splinefunktionen als*[123]

$$N_\kappa^\mu(x \,|\, T) = N_{\kappa_1}^{\mu_1}(x_1 \,|\, T_1)\ N_{\kappa_2}^{\mu_2}(x_2 \,|\, T_2)\,. \quad (6.5)$$

4. *Die Kontrollpunkte ergeben sich schließlich als* $\mathbf{d}_\kappa$, $\kappa \in \mathbb{N}^2$.

Mit all dieser schönen Notation schreibt sich unsere Splinefläche dann schön kompakt als

$$N_\mu \mathbf{d}\,(x \,|\, T) = \sum_{\kappa \le \nu} \mathbf{d}_\kappa\, N_\kappa^\mu(x \,|\, T)\,, \quad (6.6)$$

was schon fast wieder so aussieht wie der univariate Fall und sich außerdem noch sehr einfach auf eine *bliebige* Zahl von Variablen verallgemeinern lässt.

Bevor wir das tun, sehen wir uns aber erst noch schnell an, wie wir eine Tensorproduktfläche mit dem Algorithmus von de Boor auswerten: Eigentlich ist das nichts anderes als die Anwendung der Idee aus (6.1) und (6.2), indem wir zuerst für $j = 1, \dots, \nu_1$ die Koeffizienten

$$d_j(y) = \sum_{k=1}^{n} d_{jk}\, N_k^{\mu_2}(y \,|\, T_2)$$

und dann den univariaten Spline mit diesen Kontrollpunkten und der Knotenfolge T_1 an der Stelle x auswerten, siehe Abb 6.1.

Übung 6.1 Programmieren Sie den Algorithmus von de Boor zur Auswertung von funktionalen Flächen der Form $f : \mathbb{R}^2 \to \mathbb{R}$. ◊

Übung 6.2 Bestimmen Sie die *Greville-Abszissen* zu einer Tensorprodukt-Knotenfolge. Welche Struktur haben sie? ◊

6.2 Tensorprodukt in beliebig vielen Variablen

Die zweidimensionale Idee mit den „Kurven entlang Kurven" war natürlich eigentlich nur eine anschauliche Motivation für ein Konzept, das generell in beliebig vielen Variablen funktioniert.

Definition 6.2 (Allgmeine Tensorprodukte) *Für vorgegebenes* $s \in \mathbb{N}$ *definieren wir*

[123] Jetzt nutzen wir aus, daß wir die beiden Größen m_1 und m_2 in den Vektor $\mu = (\mu_1, \mu_2) \in \mathbb{N}^2$ codiert haben.

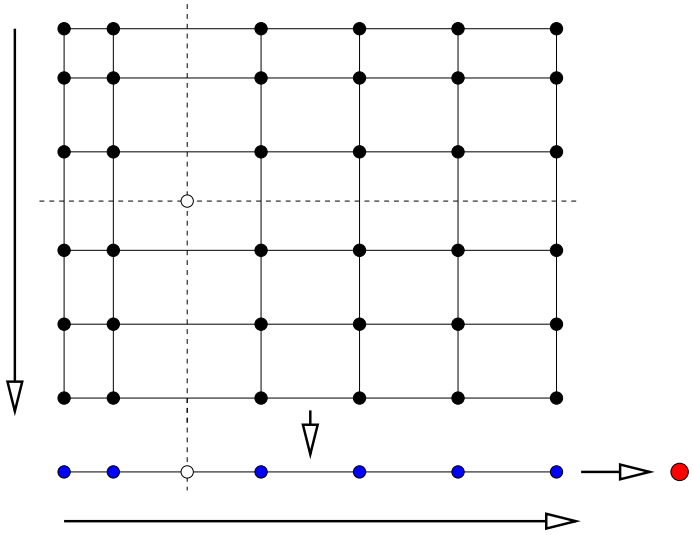


Abbildung 6.1: Der Algorithmus von de Boor für bivariate Tensorproduktflächen: Auf jede „Spalte“ von Kontrollpunkten wenden wir den univariaten Algorithmus zur Auswertung an y an und erhalten so eine „Zeile“ von Kontrollpunkten, die zur Funktion $\mathbf{f}_y$ gehören. Diese werten wir mit einem weiteren Aufruf des Algorithmus aus und schon wissen wir den Wert an der Stelle (x, y).

1. *zu Knotenfolgen*

$$T_j = \left\{ t_{j,1}, \ldots, t_{j,\nu_j+\mu_j+1} \right\}, \qquad j = 1, \ldots, s,$$

der jeweiligen Ordnung μ_j *das Tensorprodukt*

$$T = \bigotimes_{j=1}^{s} T_j := \{ t_\alpha = (t_{1,\alpha_1}, \ldots, t_{1,\alpha_s}) \ : \ \alpha \le \nu + \mu + \mathbf{1} \}.$$

2. *den* ***Tensorprodukt-B-Spline***

$$N_\kappa^\mu(x \,|\, T) = \prod_{j=1}^{s} N_{\kappa_j}^{\mu_j}(x_j \,|\, T_j). \tag{6.7}$$

3. *zu Kontrollpunkten*[124] $\mathbf{d} = [\mathbf{d}_\kappa \,:\, \kappa \le \nu]$ *die Splinekurve*

$$\mathsf{N}_\mu \mathbf{d}(\cdot \,|\, \mathsf{T}) = \sum_{\kappa \le \nu} \mathbf{d}_\kappa \, \mathsf{N}_\kappa^\mu(\cdot \,|\, \mathsf{T}) \,.$$

Auf diese Tensorproduktsplines können wir nun unsere gesamte univariate Theorie fast ohne Aufwand übertragen! Fangen wir mit einem einfachen Beispiel an.

Lemma 6.3 *Die B-Splines* N_κ^μ *bilden eine nichtnegative Teilung der Eins.*

Beweis: Die Nichtnegativität folgt unmittelbar aus (6.7), für die Teilung der Eins verwenden wir Induktion über s, wobei der Fall $s = 1$ uns ja nun wirklich bekannt sein sollte. Mit $\mu' = (\mu_1, \ldots, \mu_s)$ und entsprechend ν', x' und T' wird ansonsten

$$\begin{aligned} \sum_{\kappa \le \nu} \mathsf{N}_\kappa(x \,|\, \mathsf{T}) &= \sum_{\kappa' \le \nu'} \sum_{\kappa_s = 1}^{\nu_s} \mathsf{N}_{\kappa'}^{\mu'}(x' \,|\, \mathsf{T}') \; \mathsf{N}_{\kappa_s}^{\mu_s}(x_s \,|\, \mathsf{T}_s) \\ &= \sum_{\kappa' \le \nu'} \mathsf{N}_{\kappa'}^{\mu'}(x' \,|\, \mathsf{T}') \underbrace{\left(\sum_{\kappa_s = 1}^{\nu_s} \mathsf{N}_{\kappa_s}^{\mu_s}(x_s \,|\, \mathsf{T}_s) \right)}_{=1} \end{aligned}$$

und nach Induktionsvoraussetzung hat die gesamte Summe den Wert Eins. □

Übung 6.3 Zeigen Sie: Der Träger des B-Splines N_κ^μ ist gerade der Quader

$$\mathsf{I}_\kappa^\mu = \bigotimes_{j=1}^{s} \left[t_{j,\kappa_j}, t_{j,\kappa_j + \mu_j + 1} \right].$$

♢

Definieren wir die Vielfachheit[125] φ eines Knotens $t = t_\kappa \in \mathsf{T}$ nun als *vektorielle* Größe

$$\varphi(t) = \left(\varphi\left(t_{j,\kappa_j}\right) \,:\, j = 1, \ldots, s \right),$$

dann können wir den Splineraum $\mathbb{S}_m(\mathsf{T})$ als Menge aller stückweisen Tensorprodukt-Polynome der Ordnung μ,

$$\Pi_\mu = \operatorname{span}\{x^\kappa \,:\, \kappa \le \mu\},$$

[124] Das ist nun ein bisschen interessanter aus einer „organisatorischen" Perspektive, denn dieses Objekt ist nun eine „Matrix" der Dimension $d \times \nu = d \times \nu_1 \times \cdots \times \nu_s$. Glücklicherweise können `Matlab` und `Octave` auch mit derartigen Objekten umgehen.

[125] Das Symbol φ steht für „Fielfachheit".

definieren, bei denen an einem Knoten der Ordnung $\varphi(t)$ alle partiellen Ableitung

$$\frac{\partial^{|\alpha|}}{\partial x^{\alpha}}, \qquad \alpha \leq \mu - \varphi(t),$$

existieren und stetig sind. Und natürlich haben wir dann wieder unmittelbar unseren Curry-Schoenberg:

Die B-Splines N^{μ}_{κ}, $\kappa \leq \nu$, bilden eine Basis von $\mathbb{S}_{\mu}(T)$.

Auch die Interpolationscharakterisierung, also den Satz von Schoenberg-Whitney, könnnen wir ohne weiteres übertragen. Dazu seien $X_1, \ldots, X_s \subset \mathbb{R}$ die Projektionen der Interpolationspunkte auf die Koordinatenachsen und

$$X = \bigotimes_{j=1}^{s} X_j = \{x_{\kappa} = (x_{1,\kappa_1}, \ldots, x_{s,\kappa_s}) \ : \ \kappa \leq \nu\}.$$

Dann beweist man sehr leicht den folgenden Satz.

Satz 6.4 (Schoenberg-Whitney für Tensorprodukte) *Der Splineraum $\mathbb{S}_{\mu}(T)$ erlaubt eindeutige Interpolation bezüglich X genau dann wenn $x_{\kappa} \in I^{\mu}_{\kappa}$, $\kappa \leq \nu$.*

Übung 6.4 Beweisen Sie Satz 6.4, natürlich unter Ausnutzung von Satz 4.2. ◊

6.3 Pferdefüße

Was ist nun so schlimm an Tensorprodukt-Splines, daß wir uns überhaupt noch weitere Kapitel in diesem Skript gönnen müssen? Denn auch Glättungssplines, Knoteneinfügen und all die vielen schönen Dinge, die wir univariat gemacht haben, lassen sich ganz einfach auf Tensorprodukte übertragen. Und schon haben wir das erste Problem: Mathematisch sind Tensorprodukte nicht sonderlich aufregend, es geht alles zu glatt, die Struktur ist total univariat. Gut, aber das allein wäre natürlich ein schwaches Argument, gäbe es nicht auch wirklich substantielle Probleme, von denen wir hier ein paar auflisten wollen:

- Der Paramaterbereich ist immer ein Quader und die resultierenden Flächen haben immer eine „rechteckige“ Struktur, die aber nicht immer flexibel genug ist, denn relativ oft werden beispielsweise Flächen über Dreiecken benötigt. Dies wird von CAD-Benutzern oft dadurch behoben, daß alle Kontrollpunkte entlang einer Kante zusammenfallen, wodurch das Rechteck zum Dreieck degeneriert, aber die so entstehenden Singularitäten sind grausam und sehr schwer zu handhaben.

- Die Komplexität wächst exponentiell! Die Anzahl der Koeffizienten und damit die Dimension von $\mathbb{S}_m(T)$ ist $\prod \nu_j$, also bei „Gleichverteilung“ der Knoten in allen Variablen etwa n^s. Will man nun ein Interpolationsproblem lösen, dann enthält die Kollokationsmatrix bereits n^{2s} Einträge und ein typisches direktes Lösungsverfahren würde $O(n^{3s})$ Rechenoperationen benötigen. Wie dramatisch das wird, sieht man schon bei 10 Punkten in jeder Dimension und 6 Variablen[126], dann haben wir es mit 10^{12} Matrixeinträgen zu tun, was bereits einige Terabyte wären.

- Natürlich sind die Matrizen zu Tensorprodukt-Splines immer noch recht dünn besetzt, aber die Bandstruktur ist eine Bandstruktur *in Multiindizes*, wenn man diese linear anordnet, um die Daten in einer „normalen“ Matrix speichern zu können, dann sind die Bänder schon recht groß.

- **Alles** muss Tensorproduktstruktur haben! Zwar lässt sich der Satz von Schoenberg-Whitney sehr einfach verallgemeinern, aber die Charakterisierung gilt nur noch für eine sehr eingeschränkte Klasse von Interpolationspunktmengen, nämlich die, die die Tensorproduktstruktur $X = \bigotimes X_j$ haben, also ein Gitter bilden. Und solche Abtastungen muss man in der Realität erst einmal haben! Laserscans von 3D-Objekten liefern es jedenfalls nicht!

- Man merkt das auch beim Knoteneinfügen! Bei Tensorproduktsplines kann man keinen einzelnen Knoten in T einfügen, sondern nur in eine der Mengen T_j! In T fügt man dann die ganze Menge

  $$T^* = T_1 \otimes \cdots \otimes T_{j-1} \otimes t^* \otimes T_{j+1} \otimes \cdots \otimes T_s$$

  ein, die immerhin aus

  $$|\nu + \mu| + s - \nu_j - \mu_j - 1 = \sum_{k \neq k} (\nu_k + \mu_k + 1)$$

  Knoten besteht. In zwei Variablen, siehe Abb. Abb:deBoor2d, entspräche dies dem Einfügen einer horizontalen oder vertikalen Knotenlinie.

So, jetzt haben wir doch genug Ausreden gefunden, um uns endlich mit mathematisch schönen und anspruchsvollen multivariaten Splines allgemeinerer Natur zu befassen.

[126] Was für gewisse Anwendungen noch klein ist!

Manches sagt ich,
mehr noch wollt ich,
ließe zur Rede
Raum das Geschick.
Die Stimme weicht,
Wunden schwellen:
Wahres sprach ich;
will nun enden.

Die Edda, Das jüngere Sigurdlied

A Anhang

Hier findet sich die eine oder andere „Kleinigkeit“, die im normalen Text keinen Platz hatte, beispielsweise die Anleitung zu `lp_solve`.

```
   LP_SOLVE  Solves mixed integer linear programming problems.

SYNOPSIS: [obj,x,duals] = lp_solve(f,a,b,e,vlb,vub,xint,scalemode,keep)

   solves the MILP problem

            max v = f'*x
              a*x <> b
                vlb <= x <= vub
                x(int) are integer

ARGUMENTS: The first four arguments are required:

         f: n vector of coefficients for a linear objective function.
         a: m by n matrix representing linear constraints.
         b: m vector of right sides for the inequality constraints.
         e: m vector that determines the sense of the inequalities:
                  e(i) = -1  ==> Less Than
                  e(i) =  0  ==> Equals
                  e(i) =  1  ==> Greater Than
       vlb: n vector of lower bounds. If empty or omitted,
            then the lower bounds are set to zero.
       vub: n vector of upper bounds. May be omitted or empty.
      xint: vector of integer variables. May be omitted or empty.
 scalemode: scale flag. Off when 0 or omitted.
      keep: Flag for keeping the lp problem after it's been solved.
            If omitted, the lp will be deleted when solved.

OUTPUT: A nonempty output is returned if a solution is found:

       obj: Optimal value of the objective function.
         x: Optimal value of the decision variables.
     duals: solution of the dual problem.
```

Abbildung A.1: Beschreibung der Funktion `lp_solve` aus (Berkelaar *et al.*, 2000). Dies ist nur die „einfache“ und intuitive Schnittstelle zu `octave`, es gibt auch wesentlich subtilere und speichereffizientere Zugriffsmethoden. Aber für unsere Zwecke reicht das hier.

```
-- Function File: [XOPT, FMIN, STATUS, EXTRA] = glpk (C, A, B, LB, UB,
      CTYPE, VARTYPE, SENSE, PARAM)
    Solve a linear program using the GNU GLPK library.  Given three
    arguments, 'glpk' solves the following standard LP:

         min C'*x

    subject to

         A*x  = b
           x >= 0

    but may also solve problems of the form

         [ min | max ] C'*x

    subject to

         A*x [ "=" | "<=" | ">=" ] b
           x >= LB
           x <= UB

    Input arguments:

   C
         A column array containing the objective function coefficients.

   A
         A matrix containing the constraints coefficients.

   B
         A column array containing the right-hand side value for each
         constraint in the constraint matrix.

   LB
         An array containing the lower bound on each of the variables.
         If LB is not supplied, the default lower bound for the
         variables is zero.

   UB
         An array containing the upper bound on each of the variables.
         If UB is not supplied, the default upper bound is assumed to
         be infinite.
```

Abbildung A.2: Beschreibung der Funktion `glpk` aus (Makhorin, 2000), Teil I.

```
CTYPE
      An array of characters containing the sense of each
      constraint in the constraint matrix.  Each element of the
      array may be one of the following values
     '"F"'
           A free (unbounded) constraint (the constraint is
           ignored).

     '"U"'
           An inequality constraint with an upper bound ('A(i,:)*x
           <= b(i)').

     '"S"'
           An equality constraint ('A(i,:)*x = b(i)').

     '"L"'
           An inequality with a lower bound ('A(i,:)*x >= b(i)').

     '"D"'
           An inequality constraint with both upper and lower bounds
           ('A(i,:)*x >= -b(i)' _and_ ('A(i,:)*x <= b(i)').

VARTYPE
      A column array containing the types of the variables.
     '"C"'
           A continuous variable.

     '"I"'
           An integer variable.

SENSE
      If SENSE is 1, the problem is a minimization.  If SENSE is
      -1, the problem is a maximization.  The default value is 1.

PARAM
      A structure containing the following parameters used to
      define the behavior of solver.  Missing elements in the
      structure take on default values, so you only need to set the
      elements that you wish to change from the default.

      Integer parameters:

     'msglev ('LPX_K_MSGLEV', default: 1)'
           Level of messages output by solver routines:
          0
                No output.

          1
                Error messages only.

          2
                Normal output .

          3
                Full output (includes informational messages).
```

Abbildung A.3: Beschreibung der Funktion `glpk` aus (Makhorin, 2000), Teil II.

```
'scale ('LPX_K_SCALE', default: 1)'
      Scaling option:
     0
           No scaling.

     1
           Equilibration scaling.

     2
           Geometric mean scaling, then equilibration scaling.

'dual  ('LPX_K_DUAL', default: 0)'
      Dual simplex option:
     0
           Do not use the dual simplex.

     1
           If initial basic solution is dual feasible, use the
           dual simplex.

'price  ('LPX_K_PRICE', default: 1)'
      Pricing option (for both primal and dual simplex):
     0
           Textbook pricing.

     1
           Steepest edge pricing.

'round  ('LPX_K_ROUND', default: 0)'
      Solution rounding option:
     0
           Report all primal and dual values "as is".

     1
           Replace tiny primal and dual values by exact zero.

'itlim  ('LPX_K_ITLIM', default: -1)'
      Simplex iterations limit.  If this value is positive, it
      is decreased by one each time when one simplex iteration
      has been performed, and reaching zero value signals the
      solver to stop the search.  Negative value means no
      iterations limit.

'itcnt ('LPX_K_OUTFRQ', default: 200)'
      Output frequency, in iterations.  This parameter
      specifies how frequently the solver sends information
      about the solution to the standard output.
```

Abbildung A.4: Beschreibung der Funktion `glpk` aus (Makhorin, 2000), Teil III.

```
'branch ('LPX_K_BRANCH', default: 2)'
      Branching heuristic option (for MIP only):
     0
           Branch on the first variable.

     1
           Branch on the last variable.

     2
           Branch using a heuristic by Driebeck and Tomlin.

'btrack ('LPX_K_BTRACK', default: 2)'
      Backtracking heuristic option (for MIP only):
     0
           Depth first search.

     1
           Breadth first search.

     2
           Backtrack using the best projection heuristic.

'presol ('LPX_K_PRESOL', default: 1)'
      If this flag is set, the routine lpx_simplex solves the
      problem using the built-in LP presolver.  Otherwise the
      LP presolver is not used.

'lpsolver (default: 1)'
      Select which solver to use.  If the problem is a MIP
      problem this flag will be ignored.
     1
           Revised simplex method.

     2
           Interior point method.

'save (default: 0)'
      If this parameter is nonzero, save a copy of the problem
      in CPLEX LP format to the file '"outpb.lp"'.  There is
      currently no way to change the name of the output file.

 Real parameters:

'relax ('LPX_K_RELAX', default: 0.07)'
      Relaxation parameter used in the ratio test. If it is
      zero, the textbook ratio test is used. If it is non-zero
      (should be positive), Harris' two-pass ratio test is
      used. In the latter case on the first pass of the ratio
      test basic variables (in the case of primal simplex) or
      reduced costs of non-basic variables (in the case of
      dual simplex) are allowed to slightly violate their
      bounds, but not more than 'relax*tolbnd' or 'relax*toldj
      (thus, 'relax' is a percentage of 'tolbnd' or 'toldj''.
```

Abbildung A.5: Beschreibung der Funktion `glpk` aus (Makhorin, 2000), Teil IV.

```
'tolbnd ('LPX_K_TOLBND', default: 10e-7)'
      Relative tolerance used to check if the current basic
      solution is primal feasible.  It is not recommended that
      you change this parameter unless you have a detailed
      understanding of its purpose.

'toldj ('LPX_K_TOLDJ', default: 10e-7)'
      Absolute tolerance used to check if the current basic
      solution is dual feasible.  It is not recommended that
      you change this parameter unless you have a detailed
      understanding of its purpose.

'tolpiv ('LPX_K_TOLPIV', default: 10e-9)'
      Relative tolerance used to choose eligible pivotal
      elements of the simplex table.  It is not recommended
      that you change this parameter unless you have a
      detailed understanding of its purpose.

'objll ('LPX_K_OBJLL', default: -DBL_MAX)'
      Lower limit of the objective function.  If on the phase
      II the objective function reaches this limit and
      continues decreasing, the solver stops the search.  This
      parameter is used in the dual simplex method only.

'objul ('LPX_K_OBJUL', default: +DBL_MAX)'
      Upper limit of the objective function. If on the phase
      II the objective function reaches this limit and
      continues increasing, the solver stops the search.  This
      parameter is used in the dual simplex only.

'tmlim ('LPX_K_TMLIM', default: -1.0)'
      Searching time limit, in seconds. If this value is
      positive, it is decreased each time when one simplex
      iteration has been performed by the amount of time spent
      for the iteration, and reaching zero value signals the
      solver to stop the search.  Negative value means no time
      limit.

'outdly ('LPX_K_OUTDLY', default: 0.0)'
      Output delay, in seconds. This parameter specifies how
      long the solver should delay sending information about
      the solution to the standard output. Non-positive value
      means no delay.

'tolint ('LPX_K_TOLINT', default: 10e-5)'
      Relative tolerance used to check if the current basic
      solution is integer feasible.  It is not recommended
      that you change this parameter unless you have a
      detailed understanding of its purpose.
```

Abbildung A.6: Beschreibung der Funktion `glpk` aus (Makhorin, 2000), Teil V.

```
 Output values:

     'tolobj ('LPX_K_TOLOBJ', default: 10e-7)'
           Relative tolerance used to check if the value of the
           objective function is not better than in the best known
           integer feasible solution.  It is not recommended that
           you change this parameter unless you have a detailed
           understanding of its purpose.

XOPT
      The optimizer (the value of the decision variables at the
      optimum).

FOPT
      The optimum value of the objective function.

STATUS
      Status of the optimization.

      Simplex Method:
     180 ('LPX_OPT')
           Solution is optimal.

     181 ('LPX_FEAS')
           Solution is feasible.

     182 ('LPX_INFEAS')
           Solution is infeasible.

     183 ('LPX_NOFEAS')
           Problem has no feasible solution.

     184 ('LPX_UNBND')
           Problem has no unbounded solution.

     185 ('LPX_UNDEF')
           Solution status is undefined.
      Interior Point Method:
     150 ('LPX_T_UNDEF')
           The interior point method is undefined.

     151 ('LPX_T_OPT')
           The interior point method is optimal.
      Mixed Integer Method:
     170 ('LPX_I_UNDEF')
           The status is undefined.

     171 ('LPX_I_OPT')
           The solution is integer optimal.

     172 ('LPX_I_FEAS')
           Solution integer feasible but its optimality has not
           been proven
```

Abbildung A.7: Beschreibung der Funktion `glpk` aus (Makhorin, 2000), Teil VI.

```
      173 (‘LPX_I_NOFEAS’)
            No integer feasible solution.
       If an error occurs, STATUS will contain one of the following
       codes:

      204 (‘LPX_E_FAULT’)
            Unable to start the search.

      205 (‘LPX_E_OBJLL’)
            Objective function lower limit reached.

      206 (‘LPX_E_OBJUL’)
            Objective function upper limit reached.

      207 (‘LPX_E_ITLIM’)
            Iterations limit exhausted.

      208 (‘LPX_E_TMLIM’)
            Time limit exhausted.

      209 (‘LPX_E_NOFEAS’)
            No feasible solution.

      210 (‘LPX_E_INSTAB’)
            Numerical instability.

      211 (‘LPX_E_SING’)
            Problems with basis matrix.

      212 (‘LPX_E_NOCONV’)
            No convergence (interior).

      213 (‘LPX_E_NOPFS’)
            No primal feasible solution (LP presolver).

      214 (‘LPX_E_NODFS’)
            No dual feasible solution (LP presolver).

EXTRA
       A data structure containing the following fields:
      ‘lambda’
            Dual variables.

      ‘redcosts’
            Reduced Costs.

      ‘time’
            Time (in seconds) used for solving LP/MIP problem.

      ‘mem’
            Memory (in bytes) used for solving LP/MIP problem (this
            is not available if the version of GLPK is 4.15 or
            later).
```

Abbildung A.8: Beschreibung der Funktion `glpk` aus (Makhorin, 2000), Teil VII.

```
Example:

     c = [10, 6, 4]';
     a = [ 1, 1, 1;
          10, 4, 5;
           2, 2, 6];
     b = [100, 600, 300]';
     lb = [0, 0, 0]';
     ub = [];
     ctype = "UUU";
     vartype = "CCC";
     s = -1;

     param.msglev = 1;
     param.itlim = 100;

     [xmin, fmin, status, extra] = ...
        glpk (c, a, b, lb, ub, ctype, vartype, s, param);
```

Abbildung A.9: Beschreibung der Funktion `glpk` aus (Makhorin, 2000), Teil VIII. Diese Funktion ist zwar meiner Meinung nach etwas unhandlicher zu bedienen als `lp_solve`, hat aber den Vorteil, daß sie in den meisten `Octave`-Distributionen enthalten ist und nicht separat hinzugefügt werden muss.

Uns ist in alten mæren
wunders viel geseit
von Helden lobebæren
von grôzer arebeit

Das Nibelungenlied

A Literatur

Abramowitz, M., Stegun, I. A., editors (1972). *Handbook of mathematical functions*. Dover. 10th printing.

Berkelaar, M., Dirks, J., Eikland, K., Notebaert, P. (2000). `lp_solve`. `http://lpsolve.sourceforge.net/5.5`.

Boor, C. d. (1972). On calculating with B–splines. *J. Approx. Theory*, **6**:50–62.

Boor, C. d. (1978). *A practical guide to splines*. Springer–Verlag, New York.

Boor, C. d. (1990). *Splinefunktionen*. Lectures in Mathematics, ETH Zürich. Birkhäuser.

Butzer, P. L., Schmidt, M., Stark, E. L. (1988). Observations on the history of central B–splines. *Archive for History of Exact Sciences*, **39**(2):137–156.

Casteljau, P. d. (1985). *Formes à pôles*. Hermes, Paris.

Christensen, R. (2001). *Advanced Linear Modeling*. Springer Texts in Statistics. Springer, 2. edition.

Cox, M. G. (1972). The numerical evaluation of B–splines. *J. Inst. Math. Appl.*, **10**:134–149.

Craven, P., Wahba, G. (1979). Smoothing noisy data with spline functions: estimating the correct degree of smoothing by the method of generalized cross-validation. *Numer. Math.*, **31**:377–403.

Curry, H. B., Schoenberg, I. J. (1966). On Pólya frequency functions IV: The fundamental spline functions and their limits. *J. d'Analyse Math.*, **17**:71–107.

Dantzig, G. B. (1963). *Linear programming and extensions*. Pinceton University Press.

Farin, G. (1988). *Curves and Surfaces for Computer Aided Geometric Design*. Academic Press.

Farouki, R. T. (2007). *Pythagorean-Hodograph Curves: Algebra and Geometry Inseparable*. Geometry and Computation. Springer.

Gautschi, W. (1997). *Numerical Analysis. An Introduction*. Birkhäuser.

Golub, G., Heath, M., Wahba, G. (1979). Generalized cross-validation as a method for choosing a good ridge parameter. *Technometrics*, **21**:215–223.

Isaacson, E., Keller, H. B. (1966). *Analysis of Numerical Methods*. John Wiley & Sons.

Karlin, S. (1959). *Mathemtical Methods and Theory in Games, Programming and Economics*. Dover Phoenix Editions. Addison–Wesley. Dover Reprint 2003.

Lorentz, G. G. (1966). *Approximation of functions*. Chelsea Publishing Company.

Makhorin, A. (2000). `glpk`. `http://www.gnu.org/software/glpk/`.

Mallat, S. (1999). *A Wavelet Tour of Signal Processing*. Academic Press, 2. edition.

Marcus, M., Minc, H. (1969). *A Survey of Matrix Theory and Matrix Inequalities*. Prindle, Weber & Schmidt. Paperback reprint, Dover Publications, 1992.

Maresch, T. (2006). *Mathematik–Verknüpfung von 2D- und 3D–Punktwolken*. PhD thesis, Justus–Liebig–Universität Gießen. Supervisor: T. Sauer.

Marsden, M., Schoenberg, I. J. (1966). On variation diminishing spline approximation methods. *Mathmatica*, **8**:61–82.

Marsden, M., Schoenberg, I. J. (1988). On variation diminishing spline approximation methods. In de Boor, C., editor, *I. J. Schoenberg, Selected Papers*, volume 2 of *Contemporary Mathematics*, pages 247–268. Birkhäuser.

Micchelli, C. A. (1986). Interpolation of scattered data: distance matrices and conditionally positive definite functions. *Constr. Approx.*, **2**:11–22.

Micchelli, C. A. (1995). *Mathematical Aspects of Geometric Modeling*, volume 65 of *CBMS-NSF Regional Conference Series in Applied Mathematics*. SIAM.

Mørken, K., Reimers, M. An unconditionally convergent method for computing zeros of splines and polynomials. *Math. Comp.*.

Nocedal, J., Wright, S. J. (1999). *Numerical Optimization*. Springer Series in Operations Research. Springer.

Nürnberger, G. (1989). *Approximation by Spline Functions*. Springer–Verlag.

Ramshaw, L. (1987). Blossoming: A connect–the-dots approach to splines. Technical report, Digital Systems Research Center, Palo Alto.

Ramshaw, L. (1989). Blossoms are polar forms. *Comp. Aided Geom. Design*, **6**:352–378.

Sauer, T. (1996). Ein algorithmischer Zugang zu Polynomen und Splines. *Mathem. Semesterberichte*, **43**( ):169–189. Vortrag im *Eichstätter Kolloquium zur Didaktik der Mathematik*.

Sauer, T. (2000a). Numerische Mathematik I. Vorlesungsskript, Friedrich–Alexander–Universität Erlangen–Nürnberg, Justus–Liebig–Universität Gießen. `http://www.math.uni-giessen.de/tomas.sauer`.

Sauer, T. (2000b). Numerische Mathematik II. Vorlesungsskript, Friedrich–Alexander–Universität Erlangen–Nürnberg, Justus–Liebig–Universität Gießen. `http://www.math.uni-giessen.de/tomas.sauer`.

Sauer, T. (2002a). Approximationstheorie. Vorlesungsskript, Justus–Liebig–Universität Gießen. `http://www.math.uni-giessen.de/tomas.sauer`.

Sauer, T. (2002b). Optimierung. Vorlesungsskript, Justus–Liebig–Universität Gießen. `http://www.math.uni-giessen.de/tomas.sauer`.

Sauer, T. (2003). Digitale Signalverarbeitung. Vorlesungsskript, Justus–Liebig–Universität Gießen. `http://www.math.uni-giessen.de/tomas.sauer`.

Schoenberg, I. J. (1949a). Contributions to the problem of approximation of equidistant data by analytic functions. part A. – on the problem of of smoothing or graduation. a first class of analytic approximation formulae. *Quart. Appl. Math.*, **4**:45–99.

Schoenberg, I. J. (1949b). Contributions to the problem of approximation of equidistant data by analytic functions. part B. – on the second problem of osculatory interpolation. a second class of analytic approximation formulae. *Quart. Appl. Math.*, **4**:112–141.

Schoenberg, I. J. (1973). *Cardinal Spline Interpolation*, volume 12 of *CBMS-NSF Regional Conference Series in Applied Mathematics*. SIAM.

Schumaker, L. L. (1981). *Spline Funtions: Basic Theory*. Pure and Applied Mathematics: A Wiley–Interscience Series of Texts, Monographs and Tracts. John Wiley & Sons.

Seidel, H. P. (1989). A new multiaffine approach to B–splines. *Comp. Aided Geom. Design*, **6**:23–32.

Spellucci, P. (1993). *Numerische Verfahren der nichtlinearen Optimierung*. Internationale Schriftenreihe zu Numerischen Mathematik. Birkhäuser.

Struik, D. J. (1961). *Lectures on Classical Differential Geometry*. Addison–Wesley, 2nd edition. Dover reprint, 1988.

Index A