universität
wien

DIPLOMARBEIT

Titel der Diplomarbeit

Optimale Box-Einschließungen von NURBS Kurven

angestrebter akademischer Grad

Magistra der Naturwissenschaften (Mag. rer. nat.)

Verfasser:	Amra Smajic
Matrikel-Nummer:	0300959
Studienrichtung:	Mathematik
Betreuer:	Ao. Univ.-Prof. Dipl.-Ing. Dr. Hermann Schichl

Wien, am 05. 10. 2009

Inhaltsverzeichnis

Einleitung

Ursprünglich wurde das Wort „Splines“ als Bezeichnung für die Schiffsbeplankung in Längsrichtung des Rumpfes verwendet. In der Mathematik meint man damit Kurven und Oberflächen die von bestimmten Punkten geleitet („kontrolliert“) werden. Die häufigste Verwendung der Splines findet sich in diversen Grafik- bzw. CAD-Programmen, aber auch Querschnitte von Objekten in der 3D-Modellierung oder Bewegungspfade für Animationen können damit festgelegt werden. Durch Spline-Oberflächen werden runde, geometrische Körper geformt, die sich an ihre Kontrollpunkte schmiegen, die wiederum sehr einfache Formveränderung ermöglichen.
Deshalb werden sie oft zur Interpolation und Approximation benützt, da die stückweise Definition der Splines sie flexibler und einfacher zu handhaben macht als allgemeine Polynome. Vor allem entstehen bei der Interpolation oder der Approximation mit Splines nicht so starke Oszillationen, wie oft bei Polynomen höheren Grades.

In dieser Arbeit werden die Splines in den ersten beiden Kapiteln schrittweise eingeführt. Zuerst geht es um Basisfunktionen der Splines, die im zweiten Kapitel zu B-Spline-Kurven und Oberflächen zusammengesetzt werden.
Im Mittelpunkt der Aufmerksamkeit liegen aber sogenannte NURBS-Kurven. Diese „non uniform rational b-splines“ sind Spezialfälle der rationalen B-Spline-Kurven und werden in Kapitel 3 definiert.
Das Kapitel 4 beschreibt aus Gründen der Vollständigkeit die wichtigsten Algorithmen, die für Splines entwickelt wurden.

Das fünfte Kapitel stellt eine Einführung in die Intervallarithmetik dar. Die Implementierungen der Algorithmen rechnen zum größten Teil mit Intervallen, um die Rundungsfehler zu beherrschen und negative Effekte, die dadurch entstehen, auszuschließen. Zusätzlich wird ausführlicher das Intervall Newton Verfahren beschrieben, das im Zusammenhang mit Intervallanalysis eine große Rolle spielt.

Das Hauptresultat dieser Diplomarbeit befindet sich im vorletzten Kapitel. Das Ziel der Arbeit besteht darin, eine NURBS-Kurve von allen Seiten durch eine sogenannte Box einzuschließen. Dabei wird die Vorgangsweise, die hier gewählt worden ist,

genauer beschrieben. Schließlich werden Ergebnisse präsentiert und Sonderfälle behandelt, die beim Auswerten vorkommen können.

Im letzten Kapitel befinden sich schließlich die Implementationen. Die Vorlage für einige der Quellcodes stammen aus [8], wurden aber stark umgeschrieben und geändert, um für die in dieser Arbeit verwendeten Methoden nützlich zu sein.

1 B-Spline-Basisfunktionen

In diesem ersten Kapitel werden die Grundbausteine der Splines vorgestellt. Dazu zählen unter anderem die Knoten, der Knotenvektor und die Basisfunktionen. Zusätzlich werden noch die Eigenschaften dieser Grundelemente und deren Zusammenhang beschrieben. Im zweiten Unterkapitel behandeln wir die Ableitungen der Basisfunktionen, und schließlich wird im letzten Teilabschnitt erklärt, warum diese Funktionen Basisfunktionen heißen.

1.1 Definitionen und Eigenschaften

Bevor man die B-Spline-Funktionen definiert, ist es unbedingt erforderlich, andere Begriffe zu erklären, da die Definition der Basisfunktionen ohne diese Spline-Bauelemente nicht möglich ist.

Definition 1.1 (Knotenvektor)
Eine Menge $U = \{u_0, \dots, u_m\}$ mit $u_i \leqslant u_{i+1}$, $u_i \in \mathbb{R}$ wird **Knotenvektor** genannt. Die Elemente des Knotenvektors heißen dementsprechend **Knoten**. Der Knotenvektor U ist beschränkt und spannt ein Intervall $[u_0, u_m] = [a, b] \subseteq \mathbb{R}$ auf.
Das halboffene Teilintervall $[u_i, u_{i+1}) \subseteq \mathbb{R}$, das die Knoten aufspannen, nennt man das **Knotenintervall**. Es kann auch die Länge 0 haben, da die Knoten nicht notwendigerweise verschieden sein müssen. Tritt ein Knoten mehrfach in U auf, d.h. gilt für ein $i \in \{0, \dots, m+1\}$, dass $u_i = u_{i+k-1} < u_{i+k}$, so nennt man k die Ordnung oder die Multiplizität des Knoten u_i.
Man teilt die Knotenvektoren in „geklemmte“ und „ungeklemmte“ Knotenvektoren ein. Der Unterschied liegt darin, dass in der „geklemmten“ Variante die Endpunkte mehrfach vorkommen, in der „ungeklemmten“ jedoch nur einfach.
Außerdem teilt man die Knotenvektoren in periodische und nicht periodische Knotenvektoren ein. Der Unterschied besteht in der Tatsache ob die inneren, (unterschiedlichen) Knoten äquidistant angeordnet sind, oder nicht.

Es folgt ein Beispiel für die verschiedenen Knotenvektoren.

BEISPIEL

	geklemmt	ungeklemmt
periodisch	$U = \{0,0,0,1,2,3,4,5,6,7,7,7\}$	$U = \{2,3,4,5,6,7,8,9,10\}$
	$U = \{-2,-2,-1.5,-1,-0.5,0,0\}$	$U = \{-20,-15,-10,-5,0,5,10\}$
nicht periodisch	$U = \{0,0,0,1.8,2.5,3.3,4,6,7,7,7\}$	$U = \{-0.3,0.1,2,2.7,3.9,5.6.9.2\}$
	$U = \{1,1,1,1,1,2.5,5.3,7,7,9,9,9,9,9,9\}$	$U = \{1.5,2,2,2.8,3.1,5,5,5,7,9.2\}$

Nun kann man den Begriff der B-Spline-Basisfunktionen definieren. Sie sind stückweise Polynomfunktionen, die rekursiv nach dem Grad der Splines definiert werden.

Definition 1.2 (B-Spline-Basisfunktionen)
Sei $U := \{u_0, \ldots, u_m\}$ ein Knotenvektor. Die i-te **B-Spline-Basisfunktion** $N_{i,p}$ ist rekursiv definiert durch:
Für $p = 0$ und für $0 \leqslant i \leqslant m - p - 1$ gilt:

$$N_{i,0}(u) := \begin{cases} 1 & u_i \leqslant u < u_{i+1}, \\ 0 & \text{sonst}, \end{cases} \tag{1.1}$$

und für $p \geqslant 1$ gilt

$$N_{i,p}(u) := \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u). \tag{1.2}$$

In der Formel (1.2) können Quotienten der Form $\frac{0}{0}$ aufterten. Diese werden gleich 0 gesetzt.
Gilt zusätzlich, dass $u_{m-p} = u_{m-p+1} = \ldots = u_m$, so setzt man $N_{m-p-1,0}(u_m) = 1$.

BEISPIELE
In Abbildung 1.1 sieht man B-Spline-Basisfunktionen der Ordnung 1, 2, 3 und 4.
In der linken Spalte sind sie auf dem periodischen Knotenvektor $U_1 = \{0,0,0,1,2,3,4,5,6,6,6\}$ definiert, und in der rechten Spalte sind sie auf dem nichtperiodischen Knotenvektor $U_2 = \{0,0,0,0.5,1,2.5,2.5,4.5,6,6,6\}$ definiert.

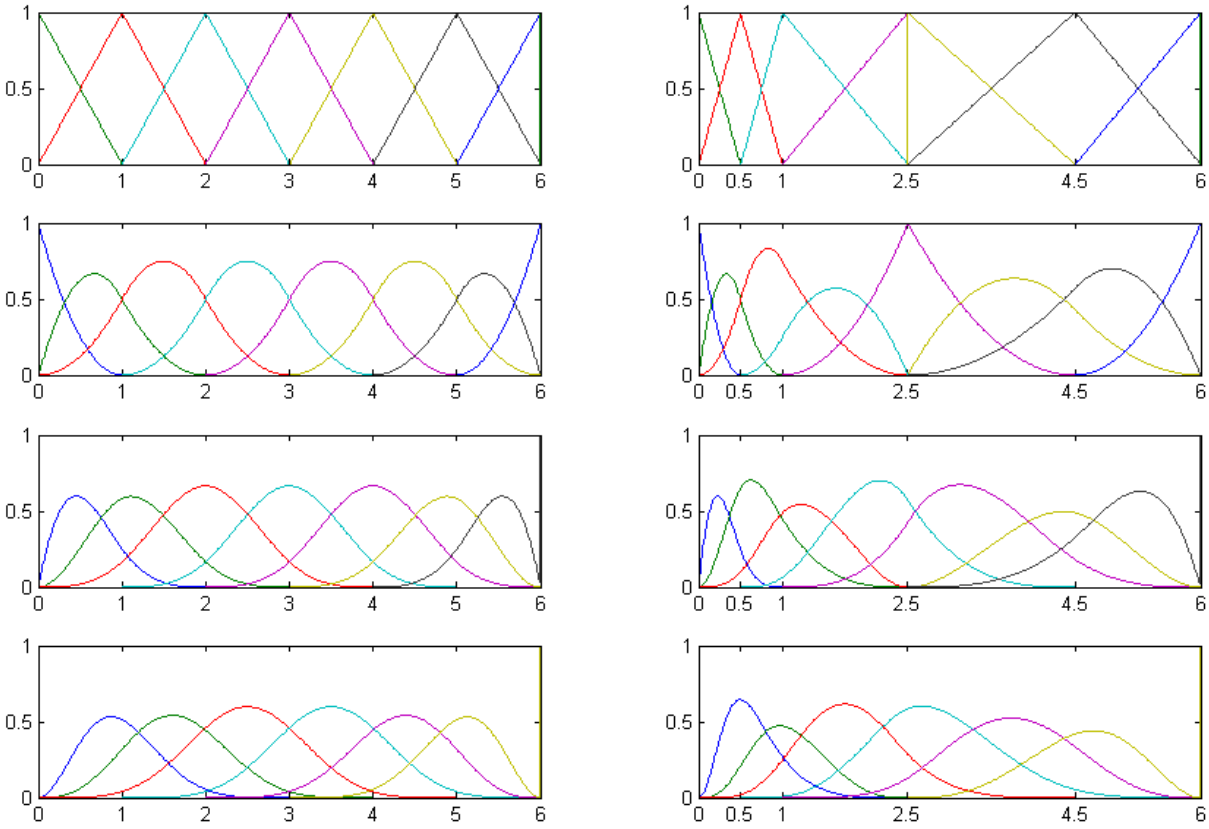


Abbildung 1.1: B-Spline-Basisfunktionen

Bemerkungen

- Die Notation wurde aus [8, ab Kapitel 2] übernommen.
- B-Spline-Basisfunktionen sind auf ganz $\mathbb{R}$ definiert. Von Interesse ist aber nur das durch den zugehörigen Knotenvektor U induzierte Intervall, da außerhalb dieses Intervalls alle $N_{i,p} \equiv 0$ sind.
- Die Anzahl der Splines auf einem Intervall wird durch die Anzahl der Knoten und durch den Grad p der Splines festgelegt. Ist $m+1$ die Anzahl der Knoten so ist $n = m - p - 1$ die Anzahl der B-Spline-Basisfunktionen vom Grad p.
- Eine B-Spline-Basisfunktion vom Grad $p \geqslant 1$ ist die Linearkombination zweier Splines vom Grad $p - 1$.

Proposition 1.1
Eigenschaften von B-Spline-Basisfunktionen

i. Jede Basisfunktion $N_{i,p}$ hat beschränkten Träger, da sie außerhalb des Intervalls $[u_i, u_{i+p+1})$ überall gleich 0 ist.

ii. Höchstens $p+1$ Basisfunktionen sind ungleich 0 auf einem einzelnen Knotenintervall $[u_i, u_{i+1})$. Das sind für $N_{i,p}(u)$ genau die Funktionen $N_{i-p,p}(u), \ldots$ $\ldots, N_{i,p}(u)$.

iii. „Nichtnegativität“: Es ist $N_{i,p}(u) \geqslant 0$ für alle i, p und für alle $u \in [u_0, u_m]$.

iv. „Partition der Eins“: Es gilt, dass $\sum_{i=0}^{p} N_{i,p}(u) = 1$ für alle $u \in U$, falls U von der Form

$$U = \{\underbrace{u_0, \ldots, u_0}_{p+1}, u_{p+1}, \ldots, u_{m-p-1}, \underbrace{u_m, \ldots, u_m}_{p+1}\}$$

ist.

v. „Regularität“: Im Inneren eines Knotenintervalls sind die B-Spline-Basisfunktionen unendlich oft stetig differenzierbar. Für $u = u_i \in U$ ist $N_{i,p}(u) \in C^{p-k}$, wobei k die Ordnung von u_i in U ist.

vi. $N_{i,p}(u)$ besitzt genau ein Maximum, außer im Fall $p = 0$.

Den Beweis zur Proposition 1.1 kann man in [8, Kaptel 2] oder in [10, Kapitel 1] nachschlagen.

1.2 Ableitungen einer B-Spline-Basisfunktion

In sehr vielen Anwendungen sind die Ableitungen der B-Spline-Basisfunktionen wichtig. Man kann diese als lineare Kombinationen von B-Spline-Basisfunktionen niedrigeren Grades darstellen bzw. man kann höhere Ableitungen als lineare Kombinationen der Ableitungen niedrigeren Grades schreiben.

Satz 1.1
Für $i = 0, \ldots, m$ und für $p \geqslant 1$ ist die Ableitung einer Basisfunktion gegeben durch

$$N'_{i,p}(u) := \frac{p}{u_{i+p} - u_i} N_{i,p-1}(u) - \frac{p}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u). \tag{1.3}$$

Beweis. Dieser Beweis wird durch Induktion nach p geführt.
Für $p = 1$ sind $N_{i,p-1}(u)$ und $N_{i+1,p-1}(u)$ entweder 0 oder 1, und somit ist $N'_{i,1}(u)$

entweder

$$\frac{1}{u_{i+1}-u_i} \qquad \text{oder} \qquad -\frac{1}{u_{i+2}-u_{i+1}}.$$

Angenommen Gleichung (1.3) ist richtig für $p-1$ mit $p \geqslant 2$. Um die Basisfunktion

$$N_{i,p}(u) = \frac{u-u_i}{u_{i+p}-u_i} N_{i,p-1}(u) + \frac{u_{i+p+1}-u}{u_{i+p+1}-u_{i+1}} N_{i+1,p-1}(u)$$

abzuleiten, benützt man die Produktregel:

$$\begin{aligned} N'_{i,p}(u) = {} & \frac{1}{u_{i+p}-u_i} N_{i,p-1}(u) + \frac{u-u_i}{u_{i+p}-u_i} N'_{i,p-1}(u) \\ & - \frac{1}{u_{i+p+1}-u_{i+1}} N_{i+1,p-1}(u) + \frac{u_{i+p+1}-u}{u_{i+p+1}-u_{i+1}} N'_{i+1,p-1}(u). \end{aligned}$$

Nun setzt man ein und erhält

$$\begin{aligned} N'_{i,p}(u) = {} & \frac{1}{u_{i+p}-u_i} N_{i,p-1}(u) - \frac{1}{u_{i+p+1}-u_{i+1}} N_{i+1,p-1}(u) \\ & + \frac{u-u_i}{u_{i+p}-u_i} \left(\frac{p-1}{u_{i+p-1}-u_i} N_{i,p-2}(u) - \frac{p-1}{u_{i+p}-u_{i+1}} N_{i+1,p-2}(u) \right) \\ & + \frac{u_{i+p+1}-u}{u_{i+p+1}-u_{i+1}} \left(\frac{p-1}{u_{i+p}-u_{i+1}} N_{i+1,p-2}(u) - \frac{p-1}{u_{i+p+1}-u_{i+2}} N_{i+2,p-2}(u) \right) \\ = {} & \frac{1}{u_{i+p}-u_i} N_{i,p-1}(u) - \frac{1}{u_{i+p+1}-u_{i+1}} N_{i+1,p-1}(u) \\ & + \frac{p-1}{u_{i+p}-u_i} \frac{u-u_i}{u_{i+p-1}-u_i} N_{i,p-2}(u) \\ & + \frac{p-1}{u_{i+p}-u_{i+1}} \left(\frac{u_{i+p+1}-u}{u_{i+p+1}-u_{i+1}} - \frac{u-u_i}{u_{i+p}-u_i} \right) N_{i+1,p-2}(u) \\ & - \frac{p-1}{u_{i+p+1}-u_{i+1}} \frac{u_{i+p+1}-u}{u_{i+p+1}-u_{i+2}} N_{i+2,p-1}(u). \end{aligned}$$

Beachtet man die Gültigkeit von

$$\begin{aligned} \frac{u_{i+p+1}-u}{u_{i+p+1}-u_{i+1}} - \frac{u-u_i}{u_{i+p}-u_i} &= \underbrace{-\frac{u_{i+p+1}-u_{i+1}}{u_{i+p+1}-u_{i+1}}}_{-1} + \frac{u_{i+p+1}-u}{u_{i+p+1}-u_{i+1}} + \underbrace{\frac{u_{i+p}-u_i}{u_{i+p}-u_i}}_{1} - \frac{u-u_i}{u_{i+p}-u_i} \\ &= \frac{u_{i+p}-u}{u_{i+p}-u_i} - \frac{u-u_{i+1}}{u_{i+p+1}-u_{i+1}}, \end{aligned}$$

so folgt weiters

$$
\begin{aligned}
N'_{i,p}(u) &= \frac{1}{u_{i+p}-u_i}N_{i,p-1}(u) - \frac{1}{u_{i+p+1}-u_{i+1}}N_{i+1,p-1}(u) \\
&\quad + \frac{p-1}{u_{i+p}-u_i}\left(\overbrace{\frac{u-u_i}{u_{i+p-1}-u_i}N_{i,p-2}(u) + \frac{u_{i+p}-u}{u_{i+p}-u_{i+1}}N_{i+1,p-2}(u)}^{N_{i,p-1}(u)}\right) \\
&\quad - \frac{p-1}{u_{i+p+1}-u_{i+1}}\left(\overbrace{\frac{u-u_{i+1}}{u_{i+p}-u_{i+1}}N_{i+1,p-2}(u) + \frac{u_{i+p+1}-u}{u_{i+p+1}-u_{i+2}}N_{i+2,p-2}(u)}^{N_{i+1,p-1}(u)}\right) \\
&= \frac{1}{u_{i+p}-u_i}N_{i,p-1}(u) - \frac{1}{u_{i+p+1}-u_{i+1}}N_{i+1,p-1}(u) \\
&\quad + \frac{p-1}{u_{i+p}-u_i}N_{i,p-1}(u) - \frac{p-1}{u_{i+p+1}-u_{i+1}}N_{i+1,p-1}(u) \\
&= \frac{p}{u_{i+p}-u_i}N_{i,p-1}(u) - \frac{p}{u_{i+p+1}-u_{i+1}}N_{i+1,p-1}(u).
\end{aligned}
$$

Damit ist die Behauptung bewiesen. □

Genauere Details zu dem Beweis und Rechenbeispiele findet man in [8, Kapitel 2].

BEISPIEL
Hier sieht man B-Spline-Basisfunktionen vom Grad 3 auf dem periodischen Knotenvektor $U = \{0, 0, 0, 0, 0.2, 0.4, 0.6, 0.8, 1, 1, 1, 1\}$ und darunter deren Ableitungen.

Für höhere Ableitungen einer B-Spline-Basisfunktion gilt die rekursive Formel

$$
N^{(k)}_{i,p}(u) := p\left(\frac{N^{(k-1)}_{i,p-1}(u)}{u_{i+p}-u_i} - \frac{N^{(k-1)}_{i+1,p-1}(u)}{u_{i+p+1}-u_{i+1}}\right). \tag{1.4}
$$

Diese Formel erhält man durch schrittweises Ableiten einer Basisfunktion. Man kann sie zu folgender Formel, die nur aus den Basisfunktionen und nicht deren Ableitungen besteht, umschreiben:

$$
N^{(k)}_{i,p}(u) = \frac{p!}{(p-k)!}\sum_{j=0}^{k} a_{k,j}N_{i+j,p-k}(u) \tag{1.5}
$$

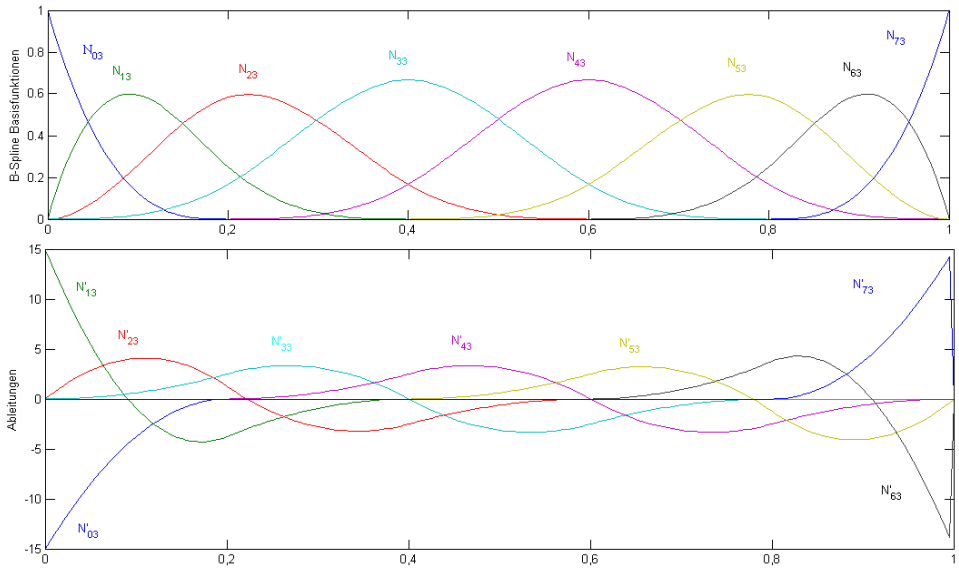


Abbildung 1.2: B-Spline-Basisfunktionen und deren Ableitungen

mit

$$
\begin{aligned}
a_{0,0} &= 1, \\
a_{k,0} &= \frac{a_{k-1,0}}{u_{i+p-k+1} - u_i}, \\
a_{k,j} &= \frac{a_{k-1,j} - a_{k-1,j-1}}{u_{i+p+j-k+1} - u_{i+j}}, \quad j = 1, \ldots, k-1, \\
a_{k,k} &= \frac{-a_{k-1,k-1}}{u_{i+p+1} - u_{i+k}}.
\end{aligned}
$$

Zu guter Letzt kann man die k-te Ableitung einer B-Spline-Basisfunktion auch als lineare Kombination k-ter Ableitungen von B-Spline-Basisfunktionen niedrigeren Grades darstellen:

$$
N_{i,p}^{(k)}(u) = \frac{p}{p-k} \left(\frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}^{(k)}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}^{(k)}(u) \right).
$$

Bei numerischen Berechnungen sollte man immer darauf achten, dass $k \leqslant p$ ist, da für jedes $k > p$ die Ableitung 0 ist.

1.3 Die Eigenschaften von Basisfunktionen

Bildet man den Vektorraum der stückweisen Polynome, so bilden die B-Spline-Basisfunktionen eine Basis dieses Raumes. Diese Tatsache wird in diesem Unterabschnitt genauer erklärt und bewiesen.

Man betrachte den Knotenvektor $\hat{U} = \{u_0, \ldots, u_k\}$ mit $u_i < u_{i+1}$ für alle $i = 0, \ldots, k-1$. Die Menge aller stückweisen Polynome vom Grad p auf $\hat{U}$ bildet einen Vektorraum $\hat{\mathcal{V}}$. Die Dimension von $\hat{\mathcal{V}}$ ist $k(p+1)$.
Weiters kann man Stetigkeitbedingungen an diese Polynome stellen. Sie sollen C^{r_i} sein an den Knoten u_i, für $i = 0, \ldots, k-1$ und $-1 \leqslant r_i \leqslant p$. (Werden für einzelne Knoten u_j keine Stetigkeitsvoraussetzungen gestellt, so setzt man $r_j = -1$.) Der entstehende Vektrorraum hat dann die Dimension

$$\dim \hat{\mathcal{V}} = k(p+1) - \sum_{i=0}^{k} (r_i + 1), \tag{1.6}$$

da jede Stetigkeitsvoraussetzung der Ordnung r_i die Dimension von $\hat{\mathcal{V}}$ um $r_i + 1$ verringert.

Die gewünschte Stetigkeit erreicht man, indem man die Ordnungen der Knoten entsprechend der Eigenschaft *v.* aus Proposition 1.1 setzt. Das heißt, man betrachtet Knotenvektoren der Form

$$U = \{\underbrace{u_0, \ldots, u_0}_{r_0}, \underbrace{u_1, \ldots, u_1}_{r_1}, \ldots, \underbrace{u_k, \ldots, u_k}_{r_k}\}$$

mit $u_i < u_{i+1}$ für alle $i = 0, \ldots, k-1$. Die Anzahl der Elemente in U beträgt $\sum_{j=0}^{k} r_j$.

Satz 1.2
Sei U wie oben und $\mathcal{V}$ der Vektorraum aller stückweisen Polynome vom Grad p, die C^{r_i} an den Knoten u_i sind.
Die B-Spline-Basisfunktionen $N_{i,p}$ bilden eine Basis des Vektorraums $\mathcal{V}$.

Beweis. Sei

$$m = -1 + \sum_{j=0}^{k} t_j.$$

Dann gibt es klarerweise genau m B-Spline-Basisfunktionen vom Grad 0 auf U

mit $N_{i,0} \not\equiv 0$, welche die gewünschten Stetigkeitvoraussetzungen erfüllen. Weiters gibt es $m-1$ B-Spline-Basisfunktionen vom Grad 1 auf U mit $N_{i,1} \not\equiv 0$. Allgemein gibt es $m-p$ B-Spline-Basisfunktionen vom Grad p auf U mit $N_{i,p} \not\equiv 0$, welche ebenfalls die gewünschten Stetigkeitvorraussetzungen erfüllen. Laut Eigenschaft *v.* aus Proposition 1.1 gilt $t_i = p - r_i$, wobei r_i die Stetigkeitsvoraussetzung an dem Knoten u_i für $i = 0, \ldots, k$ ist. Setzt man dies in die Gleichung (1.6) ein, so ergibt sich

$$\begin{aligned}
\dim \mathcal{V} &= k(p+1) - \sum_{j=0}^{k} (p - t_j + 1) \\
&= k(p+1) - (k+1)p + \sum_{j=0}^{k} t_j - (k+1) \\
&= -p + 1 + \sum_{j=0}^{k} t_j \\
&= m - p.
\end{aligned}$$

Somit entspricht die Dimension von $\mathcal{V}$ der Anzahl der B-Spline-Basisfunktionen. Es bleibt noch zu zeigen, dass die Basisfunktionen linear unabhängig sind. Dies beweist man wiederum durch Induktion nach p.
Die Basisfunktionen für $p = 0$ sind linear unabhängig, da sie laut Definition disjunkte Träger haben.
Angenommen für $p \geqslant 1$ seien die B-Spline-Basisfunktionen vom Grad $p-1$ linear unabhängig. Man setzt $n = m - p - 1$ und muss zeigen, dass

$$\sum_{i=0}^{n} \alpha_i N_{i,p}(u) = 0 \quad \text{für alle } u \text{ ist.}$$

Mit Gleichung (1.3) folgt

$$\begin{aligned}
0 = \left(\sum_{i=0}^{n} \alpha_i N_{i,p}(u) \right)' &= \sum_{i=0}^{n} \alpha_i N'_{i,p}(u) \\
= p \sum_{i=0}^{n} \alpha_i \left(\frac{N_{i,p-1}(u)}{u_{i+p} - u_i} - \frac{N_{i+1,p-1}(u)}{u_{i+p+1} - u_{i+1}} \right).
\end{aligned}$$

Es gilt weiters, dass

$$\begin{aligned}
0 &= \sum_{i=0}^{n} \alpha_i \frac{N_{i,p-1}(u)}{u_{i+p} - u_i} - \sum_{i=0}^{n} \alpha_i \frac{N_{i+1,p-1}(u)}{u_{i+p+1} - u_{i+1}} \\
&= \sum_{i=0}^{n} \alpha_i \frac{N_{i,p-1}(u)}{u_{i+p} - u_i} + \sum_{i=1}^{n+1} \alpha_{i-1} \frac{N_{i,p-1}(u)}{u_{i+p} - u_i}.
\end{aligned}$$

Nun ist $N_{0,p-1(u)} = N_{n+1,p-1}(u) = 0$, und durch die Änderung der Indizierung im zweiten Term ergibt sich die Gleichung

$$0 = \sum_{i=1}^{n} \frac{\alpha_i - \alpha_{i-1}}{u_{i+p} - u_i} N_{i,p-1}(u).$$

Wegen der Induktionsannahme gilt $\alpha_i - \alpha_{i-1} = 0$ für alle i, womit gilt, dass alle α_i gleich sein müssen. Nun ist noch zu zeigen dass die $\alpha_i = 0$ sein müssen. Sei dazu $u \in (u_j, u_{j+1})$.
Es ist

$$\sum_{i=j-p}^{j} \alpha_i \underbrace{N_{i,p}(u)}_{\geqslant 0} = 0.$$

Aber mindestens eine der Funktionen ist größer als 0:

$$N_{j,p}(u) = \underbrace{\frac{u - u_j}{u_{j+p} - u_j}}_{k_0 > 0} N_{j,p-1} + \frac{u_{j+p+1} - u}{u_{j+p+1} - u_j} \underbrace{N_{j+1,p-1}(u)}_{=0} =$$

$$k_0 N_{j,p-1}(u) = k_0 \left(\underbrace{\frac{u - u_j}{u_{j+p-1} - u_j}}_{k_1 > 0} N_{j,p-2} + \frac{u_{j+p} - u}{u_{j+p} - u_j + 1} \underbrace{N_{j+1,p-2}(u)}_{=0} \right) =$$

$$k_0 k_1 N_{j,p-2}(u) = \ldots = \underbrace{k_0 \cdot \ldots \cdot k_{p-1}}_{>0} \underbrace{N_{j_0}}_{=1}.$$

Damit muss α_j Null sein und somit auch alle anderen α_i.

□

2 B-Spline-Kurven und Oberflächen

Mit Hilfe der Spline-Basisfunktionen aus dem ersten Kapitel werden in diesem Kapitel B-Spline-Kurven beschrieben. Es wird ferner auf deren Eigenschaften verwiesen, und es werden Methoden vorgestellt, um Ableitungen der B-Spline-Kurven zu bilden.
Danach werden B-Spline-Oberflächen und ihre Eigenschaften vorgestellt. Ebenso werden Formeln angegeben, um partielle Ableitugen dieser Oberflächen zu berechnen.
Zum Schluss dieses Kapitels wird noch ein wichtiger Spezialfall der Splines vorgestellt, die Bézier Kurven.

2.1 Definition und Eigenschaften von B-Spline-Kurven

Spline-Kurven sind stetige Kurven, deren Verlauf durch eine Menge geordneter Punkte in der Ebene oder im Raum vorgegeben werden. Zusätzlich haben sie aufgrund ihrer speziellen Konstruktionsweise eine Reihe von interessanten Eigenschaften, die das Arbeiten mit solchen Kurven erleichtern.

Definition 2.1 (Spline-Kurve)
Eine **B-Spline-Kurve** vom Grad p ist definiert durch

$$C(u) = \sum_{i=0}^{n} N_{i,p}(u) P_i \qquad a \leqslant u \leqslant b. \tag{2.1}$$

Dabei nennt man P_i für $0 \leqslant i \leqslant n$ die Kontrollpunkte und das Polygon, das sie bilden, Kontrollpolygon. $N_{i,p}(u)$ sind B-Spline-Basisfunktionen vom Grad p, die auf dem nicht notwendigerweise periodischen geklemmten Knotenvektor

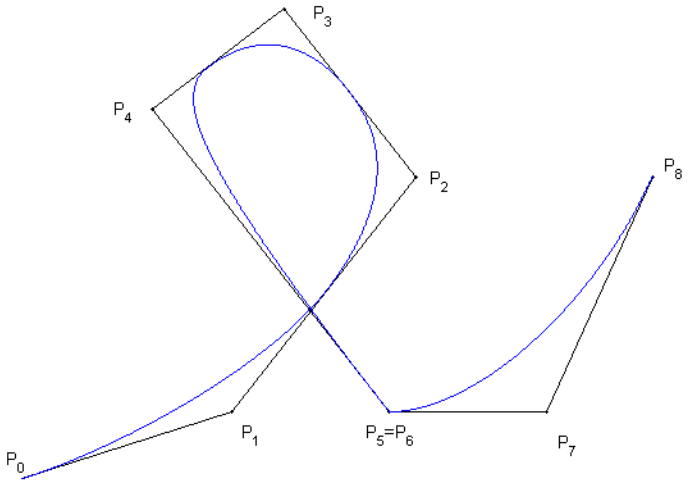
$$U = \{\underbrace{a, \ldots, a}_{p+1}, u_{p+1}, \ldots, u_{m-p-1}, \underbrace{b, \ldots, b}_{p+1}\}$$

definiert sind. U besitzt also $m+1$ Knoten, die auch mit $u_0 = a, u_1 = a, \ldots, u_p = a$, $u_{p+1}, \ldots, u_{m-p-1}, u_{m-p} = b, \ldots, u_m = b$ bezeichnet werden. Die inneren Punkte u_i

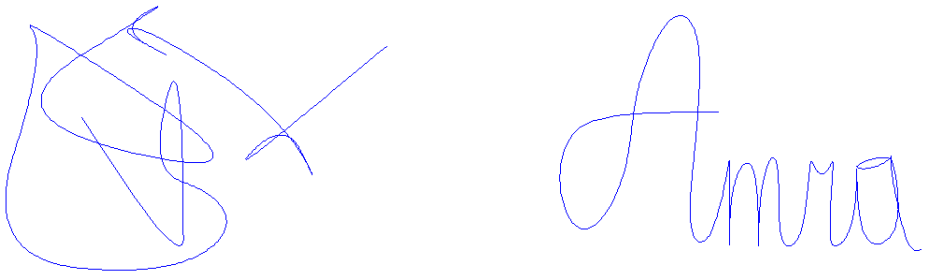
für $i = 0, \ldots, m-1$ in U dürfen hier durchaus mehrfach vorkommen.

BEISPIELE

1. B-Spline-Basisfunktionen der Ordnung 2 auf dem Knotenvektor $U_1 = \{0,0,0,1,2,3,4,4,5,5,5\}$.



2. Eine etwas längere Kurve und ein weiteres Beispiel.



Proposition 2.1
Eigenschaften von B-Spline-Kurven

i. $C(u)$ ist eine stückweise Kurve, deren polynomiale Stücke $C_i(u)$ für $i = 0, \ldots, n-1$ Polynome vom Grad p sind. Der Zusammenhang zwischen der Anzahl der Knoten $(m+1)$, der Anzahl der Kontrollpunkte $(n+1)$ und des Grades der

B-Spline-Basisfunktionen p ist durch folgende Gleichung gegeben:

$$m = n + p + 1. \tag{2.2}$$

ii. Die B-Spline-Kurven interpolieren ihre Endpunkte P_0 und P_n. Insbesondere ist $C(a) = P_0$ und $C(b) = P_n$.

iii. B-Spline-Kurven sind invariant unter affinen Transformationen (Translationen, Rotationen und Skalierungen). Die Transformationen werden nur auf die Kontrollpunkte P_i des Kontrollpolygons angewandt.

iv. Die B-Spline-Kurve ist vollständig in der konvexen Hülle ihres Kontrollpolygons enthalten. Genauer ist für alle u in einem Knotenintervall $[\hat{u}_i, \hat{u}_{i+1})$ das zugehörige Kurvenstück $C_i(u)$ (für $u \in [\hat{u}_i, \hat{u}_{i+1})$) in der konvexen Hülle von $P_{i-p}, \ldots, P_i$ enthalten.

v. Wenn man einen Kontrollpunkt P_i ändert, so hat dies nur im Intervall $[\hat{u}_i, \hat{u}_{i+p+1})$ Auswirkungen auf die Kurve $C(u)$.

vi. Das Kontrollpolygon bildet eine stückweise lineare Approximation an die Kurve. Dabei gilt die Regel: Je niedrieger der Grad p der Kurve, desto besser wird sie durch das Polygon approximiert. Für $p = 1$ sind das Polygon und die Kurve identisch.

vii. Im $\mathbb{R}^2$ gilt die Tatsache, dass keine Gerade mehr Schnittpunkte mit der Kurve selbst als mit ihrem Kontrollpolygon hat. Im$\mathbb{R}^3$ gilt diese Aussage für jede Ebene.

viii. Weil eine B-Spline-Kurve eine lineare Kombination aus den B-Spline-Basisfunktionen ist, hat sie auch dieselben Regularitätseigenschaften. Genauer gesagt ist sie im Inneren eines jeden Knotenintervalls unendlich oft differenzierbar, und für $u = \hat{u}_i \in U$ ist $C(u)$ höchstens $(p - t_i)$-mal differenzierbar, wenn t_i die Ordnung des Knotens $\hat{u}_i$ für $i = 0, \ldots, k$ ist.

Genauere Dateils und Beweise für diese Eigenschaften findet man in [10, Kapitel 1] oder in [8, Kapitel 3].

2.2 Ableitungen einer B-Spline-Kurve

Weil jede B-Spline-Kurve mit Hilfe von Linearkombinationen von B-Spline-Basisfunktionen erzeugt wird, erhält man ihre Ableitung durch die Ableitung der Basisfunktionen.

Genauer ist für eine B-Spline-Kurve

$$C(u) = \sum_{i=0}^{n} N_{i,p}(u) P_i,$$

die auf dem Knotenvektor U definiert ist, die Ableitung gegeben durch

$$C'(u) = \sum_{i=0}^{n} N'_{i,p}(u) P_i. \tag{2.3}$$

Diese Formel kann man folgendermaßen umschreiben:

$$\begin{aligned}
C'(u) &= \sum_{i=0}^{n} N'_{i,p}(u) P_i = \sum_{i=0}^{n} \left(\frac{p}{u_{i+p} - u_i} N_{i,p-1}(u) - \frac{p}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u) \right) P_i \\
&= \left(p \sum_{i=-1}^{n-1} N_{i+1,p-1}(u) \frac{P_{i+1}}{u_{i+p+1} - u_{i+1}} \right) - \left(p \sum_{i=0}^{n} N_{i+1,p-1}(u) \frac{P_i}{u_{i+p+1} - u_{i+1}} \right) \\
&= p \underbrace{\frac{N_{0,p-1}(u) P_0}{u_p - u_0}}_{=0 \text{ weil } u_0 = u_p} + p \sum_{i=0}^{n-1} N_{i+1,p-1}(u) \frac{P_{i+1} - P_i}{u_{i+p+1} - u_{i+1}} - p \underbrace{\frac{N_{n+1,p-1}(u) P_n}{\underbrace{u_{n+p+1}}_{=u_m} - \underbrace{u_{n+1}}_{=u_{m-p}}}}_{=0} \\
&= p \sum_{i=0}^{n} N_{i+1,p-1}(u) \frac{P_i}{u_{i+p+1} - u_{i+1}}.
\end{aligned}$$

Nun betrachtet man den Knotenvektor

$$U' = \{ \underbrace{u_0}_{=a}, \underbrace{u_1}_{=a} \ldots, \underbrace{u_{p-1}}_{=a}, u_p \ldots, u_{m-p}, \underbrace{u_{m-p+1}}_{=b}, \ldots, \underbrace{u_{m-2}}_{=b}, \underbrace{u_{m-1}}_{=b} \}.$$

($U' = U \setminus \{u_0, u_m\}$ und die Ordnung der Endpunkte a und b ist nun jeweils p; somit hat U' zwei Knoten weniger als U, also $m-1$ Elemente.) Die B-Spline-Basisfunktion $N_{i+1,p-1}(u)$ auf dem Knotenvektor U ist gleich der B-Basisfunktion $N_{i,p-1}(u)$ auf dem Knotenvektor U'.
Setzt man weiters

$$Q_i = p \frac{P_{i+1} - P_i}{u_{i+p+1} - u_{i+1}}, \tag{2.4}$$

so ist die Formel auf dem Knotenvektor U' für die Ableitung der B-Spline-Kurve $C(u)$ auf dem Knotenvektor U gegeben durch

$$C'(u) = \sum_{i=0}^{n-1} N_{i,p-1}(u) Q_i \tag{2.5}$$

auf dem Knotenvektor U'. Diese Formel ist numerisch weniger aufwändig als die allgemeine Formel (2.3), da man die Ableitungen der Basisfunktionen nicht zu be-

rechnen braucht. Außerdem ist die Anzahl der Basisfunktionen geringer, da der Knotenvektor U' weniger Elemente besitzt als U.

Die Ableitungen einer B-Spline-Kurve höheren Grades können durch folgende rekursive Methode berechnet werden:

$$\begin{aligned} C(u) = C^{(0)}(u) &= \sum_{i=0}^{n} N_{i,p}(u) P_i^{(0)} \quad \text{auf dem Knotenvektor } U = U^{(0)} \\ C'(u) = C^{(1)}(u) &= \sum_{i=0}^{n-1} N_{i,p-1}(u) P_i^{(1)} \quad \text{auf dem Knotenvektor } U^{(1)} \\ C''(u) = C^{(2)}(u) &= \sum_{i=0}^{n-2} N_{i,p-2}(u) P_i^{(2)} \quad \text{auf dem Knotenvektor } U^{(2)}. \end{aligned}$$

Schließlich ergibt sich die Formel

$$C^{(k)}(u) = \sum_{i=0}^{n-k} N_{i,p-k}(u) P_i^{(k)} \tag{2.6}$$

mit

$$P_i^{(k)} = \begin{cases} P_i & k = 0 \\ \frac{p-k+1}{u_{i+p+1}-u_{i+k}} (P_{i+1}^{(k-1)} - P_i^{(k-1)}) & k > 0 \end{cases} \tag{2.7}$$

auf dem Knotenvektor $U^{(k)} = U^{(k-1)} \setminus \{\text{Endpunkte}\}$ für $p \geqslant k \geqslant 1$.

Satz 2.1
Im Allgemeinen geht die B-Spline-Kurve C nicht durch die Punkte P_i. Es gilt jedoch, dass $C(a) = 0$ und $C(b) = 1$ ist, falls der Knotenvektor U von der Form

$$U = \{\underbrace{a, \ldots, a}_{p+1}, u_{p+1}, u_{p+2}, \ldots, u_{m-p-1}, \underbrace{b, \ldots, b}_{p+1}\}$$

ist. In diesem Fall ist die Kurve in P_0 und P_n tangential zu den Seiten des Kontrollpolygons.

Beweis. Zuerst zeigt man, dass $C(a) = P_0$.
Der Knoten a liegt im Knotenintervall $[u_p, u_p+1)$ für $p = 0$. Laut Definition 1.2 ist für $p = 0$ nur $N_{p,0}(a) = 1$. Für $i = 1, \ldots, p-1, p+1, \ldots, n$ ist $N_{i,0}(a) = 0$. Für $p > 0$ sind laut Eigenschaft *ii.* aus Proposition 1.1 höchstens $p+1$ Basisfunktionen auf $[u_p, u_p+1)$ ungleich 0, und zwar genau die Basisfunktionen $N_{0,p}(a), \ldots, N_{p,p}(a)$. Diese muss

man berechnen:

$$N_{0,p}(a) = \underbrace{\frac{a - \overbrace{u_0}^{a}}{u_p - u_0} N_{0,p-1}(a)}_{0} + \underbrace{\frac{u_{p+1} - a}{u_{p+1} - \underbrace{u_1}_{a}} N_{1,p-1}(a)}_{1} = N_{1,p-1}(a) =$$

$$= \underbrace{\frac{a - \overbrace{u_1}^{a}}{u_{p+1} - u_1} N_{1,p-2}(a)}_{0} + \underbrace{\frac{u_{p+2} - a}{u_{p+2} - \underbrace{u_2}_{a}} N_{2,p-2}(a)}_{1} = N_{2,p-2}(a) =$$

$$= \ldots = N_{3,p-3}(a) = \ldots = N_{p,p-p}(a) = N_{p,0}(a) = 1,$$

$$N_{1,p}(a) = \underbrace{\frac{a - \overbrace{u_1}^{a}}{u_{p+1} - u_1} N_{1,p-1}(a)}_{0} + \underbrace{\frac{u_{p+2} - a}{u_{p+2} - \underbrace{u_2}_{a}} N_{2,p-1}(a)}_{1} = N_{2,p-1}(a) =$$

$$= \underbrace{\frac{a - \overbrace{u_2}^{a}}{u_{p+2} - u_2} N_{2,p-2}(a)}_{0} + \underbrace{\frac{u_{p+2} - a}{u_{p+2} - \underbrace{u_3}_{a}} N_{3,p-2}(a)}_{1} = N_{3,p-2}(a) =$$

$$= \ldots = N_{4,p-3}(a) = \ldots = N_{p+1,p-p}(a) = N_{p+1,0}(a) = 0,$$

$$N_{2,p}(a) = \underbrace{\frac{a - \overbrace{u_2}^{a}}{u_{p+2} - u_2} N_{2,p-1}(a)}_{0} + \underbrace{\frac{u_{p+3} - a}{u_{p+3} - \underbrace{u_3}_{a}} N_{3,p-1}(a)}_{1} = N_{3,p-1}(a) =$$

$$= \ldots = N_{4,p-3}(a) = \ldots = N_{p+2,p-p}(a) = N_{p+2,0}(a) = 0,$$

$$\begin{aligned}
N_{3,p}(a) &= \quad \ldots = N_{4,p-1}(a) = \ldots = N_{5,p-2}(a) = \ldots = N_{p+3,0}(a) = 0,\\
N_{4,p}(a) &= \quad \ldots = N_{5,p-1}(a) = \ldots = N_{6,p-2}(a) = \ldots = N_{p+4,0}(a) = 0,\\
&\ldots\\
N_{p-1,p}(a) &= \quad \ldots = N_{2p-1,0}(a) = 0,\\
N_{p,p}(a) &= \quad \ldots = N_{2p,0}(a) = 0.
\end{aligned}$$

Schließlich setzt man das in Gleichung (2.1) ein und erhält

$$C(a) = \sum_{i=0}^{n} N_{i,p}(a)P_i = \sum_{i=0}^{p} N_{i,p}(a)P_i = \underbrace{N_{0,p}(a)}_{1} P_0 + \underbrace{N_{1,p}(a)}_{0} P_1 + \ldots + \underbrace{N_{p,p}(a)}_{0} P_p = P_0$$

Nun muss man noch $C(b) = P_n$ zeigen.
Der letzte Knoten, b, liegt im abgeschlossenen Teilintervall $[u_{m-p-1}, u_{m-p}]$, somit sind nach Eigenschaft *ii.* aus Proposition 1.1 die Basisfunktionen $N_{m-2p-1,p}(b), \ldots, \ldots, N_{m-p-1}(b)$ zu berechnen. Zusätzlich gilt laut Definition 1.2, dass $N_{m-p-1,0}(b) = 1$ ist. Alle anderen Basisfunktionen vom Grad $p = 0$ an der Stelle b sind 0.
Nun muss man die letzten $p + 1$ Basisfunktionen berechnen.

$$\begin{aligned}
N_{m-p-1,p}(b) &= \underbrace{\frac{b - u_{m-p-1}}{\underbrace{u_{m-1}}_{b} - u_{m-p-1}}}_{1} N_{m-p-1,p-1}(b) + \underbrace{\frac{\overbrace{u_m}^{b} - b}{u_m - u_{m-p}}}_{0} N_{m-p,p-1}(b) \\
&= N_{m-p-1,p-1}(b) = \\
&= \underbrace{\frac{b - u_{m-p-1}}{\underbrace{u_{m-1}}_{b} - u_{m-p-1}}}_{1} N_{m-p-1,p-2}(b) + \underbrace{\frac{\overbrace{u_m}^{b} - b}{u_m - u_{m-p}}}_{0} N_{m-p,p-2}(b) \\
&= N_{m-p-1,p-2}(b) = \ldots = N_{m-p-1,p-p}(b) = N_{m-p-1,0}(b) = 1, \\
N_{m-p-2,p}(b) &= \underbrace{\frac{b - u_{m-p-2}}{\underbrace{u_{m-2}}_{b} - u_{m-p-2}}}_{1} N_{m-p-2,p-1}(b) + \underbrace{\frac{\overbrace{u_{m-1}}^{b} - b}{u_{m-1} - u_{m-p}}}_{0} N_{m-p-1,p-1}(b) \\
&= N_{m-p-2,p-1}(b) = \\
&= \underbrace{\frac{b - u_{m-p-2}}{\underbrace{u_{m-1}}_{b} - u_{m-p-2}}}_{1} N_{m-p-2,p-2}(b) + \underbrace{\frac{\overbrace{u_{m-1}}^{b} - b}{u_{m-1} - u_{m-p-1}}}_{0} N_{m-p-1,p-2}(b) \\
&= N_{m-p-2,p-2}(b) = \ldots = N_{m-p-2,p-3}(b) = \ldots = N_{m-p-2,0}(b) = 0,
\end{aligned}$$

$$\begin{aligned}
N_{m-p-3,p}(b) &= \ldots = N_{m-p-3,0}(b) = 0, \\
&\ldots \\
N_{m-2p,p}(b) &= \ldots = N_{m-2p,0}(b) = 0, \\
N_{m-2p-1,p}(b) &= \ldots = N_{m-2p-1,0}(b) = 0.
\end{aligned}$$

Es ist laut Definition $n = m - p - 1$; somit gilt:

$$C(b) = \sum_{i=0}^{n} N_{i,p}(b)P_i = \sum_{i=m-2p-1}^{m-p-1} N_{i,p}(a)P_i = \sum_{i=n-p}^{n} N_{i,p}(a)P_i =$$
$$= \underbrace{N_{n-p,p}(a)}_{0} P_{n-p} + \underbrace{N_{n-p+1,p}(a)}_{0} P_{n-p+1} + \ldots + \underbrace{N_{n,p}(a)}_{1} P_n = P_n.$$

Es bleibt zu zeigen, dass die Tangenten an den Endpunkten der Kurve den Seiten des Kontrollpolygons entsprechen. Das bedeutet, es ist zu zeigen, dass $C'(a) = \lambda(P_0 - P_1)$ für ein $\lambda \in \mathbb{R}$.

$$C'(a) = \sum_{i=0}^{n} N'_{i,p}(a)P_i = \sum_{i=0}^{p} N'_{i,p}(a)P_i \overbrace{=}^{\text{Satz 1.1}}$$
$$= \sum_{i=0}^{p} p\left(\frac{N_{i,p-1}(a)}{u_{i+p} - u_i} - \frac{N_{i+1,p-1}(a)}{u_{i+p+1} - u_{i+1}}\right) P_i$$

Nach dem ersten Teil des Beweises gilt dass $N_{1,p-1}(a) = 1$, und für alle $i \neq 1$ gilt $N_{i,p-1}(a) = 0$. Somit ist

$$C'(a) = p\frac{N_{1,p-1}(a)P_1}{u_{p+1} - u_1} - p\frac{N_{1,p-1}(a)P_0}{u_{p+1} - u_1} = p\frac{P_1}{u_{p+1} - u_1} - p\frac{P_0}{u_{p+1} - u_1} =$$
$$= \frac{p}{u_{p+1} - u_1}(P_0 - P_1).$$

Analog zeigt man, dass $C'(b) = \nu(P_{n-1} - P_n)$ für ein $\nu \in \mathbb{R}$.

□

2.3 Definition und Eigenschaften von B-Spline-Oberflächen

Nicht nur Kurven, auch Oberflächen kann man mithilfe von B-Spline-Basisfunktionen konstruieren. Diese Oberflächen haben ebenfalls eine Reihe „schöner“ Eigenschaften. Es existieren einige Möglichkeiten, eine Oberfläche im Raum mathematisch zu beschreiben. Für die Theorie der B-Splines eignet sich dabei am besten die Darstellung als Tensorprodukt.
Diese Methode erfordert Basisfunktionen in zwei Variablen, die als Produkt von eindimensionalen Basisfunktionen in einer Variable konstruiert werden. Zusätzlich benötigt man geometrische Elemente, die in einem bidirektionalen Gitter angeordnet werden. Für B-Spline-Oberflächen sind dies die Punkte $P_{i,j}$ des Kontrollpolygons.

Definition 2.2 (B-Spline-Oberfläche)
Um B-Spline-Oberflächen zu konstruieren, benötigt man zwei nicht notwendigerweise periodische Knotenvektoren

$$U = \{\underbrace{u_0}_{=a}, \underbrace{u_1}_{=a}, \ldots, \underbrace{u_p}_{=a}, u_{p+1}, \ldots, u_{r-p-1}, \underbrace{u_{r-p}}_{=b}, \ldots, \underbrace{u_r}_{=b}\} \quad \text{und} \tag{2.8}$$

$$V = \{\underbrace{v_0}_{=c}, \underbrace{v_1}_{=c}, \ldots, \underbrace{v_p}_{=c}, v_{p+1}, \ldots, v_{s-q-1}, \underbrace{v_{s-q}}_{=d}, \ldots, \underbrace{v_s}_{=d}\}. \tag{2.9}$$

Dabei gilt: U besitzt $r+1$ Knoten mit Endpunkten a und b der Ordnung $p+1$, und V besitzt $s+1$ Elemente mit Endpunkten c und d der Ordnung $q+1$.
Auf U werden B-Spline-Basisfunktionen $N_{i,p}(u)$ vom Grad p mit $i = 0, \ldots, r-p-1$ definiert. Die Gleichung (2.2) für B-Spline-Kurven entspricht nun der Gleichung

$$r = n + p + 1.$$

Auf V werden B-Spline-Basisfunktionen $N_{j,q}(u)$ vom Grad q mit $j = 0, \ldots, s-q-1$ definiert. Die Gleichung (2.2) für B-Spline-Kurven entspricht nun der Gleichung

$$s = m + q + 1.$$

Die Kontrollpunkte $P_{i,j}$ des Kontrollpolygons (i, j siehe oben) werden bidirektional in einem Gitter angeordnet.
Die **B-Spline-Oberfläche**, die durch U, V und $P_{i,j}$ definiert wird, ist gegeben durch

$$S(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} N_{i,p}(u) N_{j,q}(v) P_{i,j}. \tag{2.10}$$

BEISPIEL
In der Abbildung 2.1 wird eine B-Spline-Oberfläche vom Grad $p = 2$ auf $U = \{0,0,0,1,2,3,4,4,5,5,5\}$ und vom Grad $q = 3$ auf $V = \{0,0,0,0,1,2,3,3,3,4,5,6,6,6,6\}$ dargestellt.

Proposition 2.2
Eigenschaften von B-Spline-Oberflächen Dadurch, dass jede B-Spline-Oberfläche als Tensorprodukt zweier B-Spline-Basisfunktionen dargestellt ist, besitzen diese Produkte die entsprechenden Eigenschaften der Basisfunktionen.

i. $N_{i,p}(u) N_{j,q}(v) \geqslant 0$ für alle i, j, p, q, u und v.

ii. Partition der Eins: $\sum_{i=0}^{n} \sum_{m}^{j=0} N_{i,p}(u) N_{j,q}(v) = 1$ für alle (u,v) in $U \times V$.

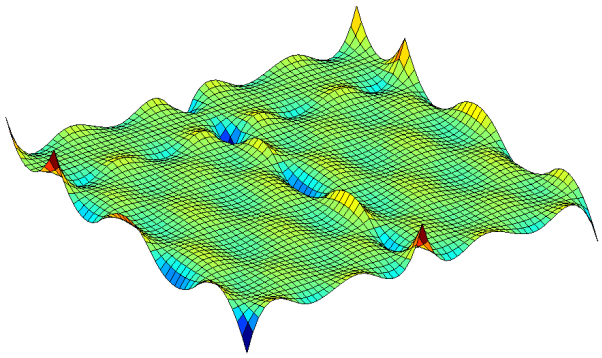


Abbildung 2.1: Eine B-Spline-Oberfläche

iii. Für $(u,v) \notin [u_i, u_{i+p+1}) \times [v_j, v_{j+q+1})$ ist $N_{i,p}(u)N_{j,q}(v) = 0$.

iv. Sind $p > 0$ und $q > 0$, so besitzt $N_{i,p}(u)N_{j,q}(v) = 1$ genau ein Maximum.

v. Es sind höchstens $(p+1)(q+1)$ der $N_{i,p}(u)N_{j,q}(v)$ auf dem Rechteck $[u_i, u_{i+1}) \times [v_j, v_{j+1})$ ungleich 0.

vi. Im Inneren der Rechtecke $[u_i, u_{i+1}) \times [v_j, v_{j+1})$ für $i = 0, \ldots, r-p-1$ und für $j = 0, \ldots, s-q-1$ existieren alle partiellen Ableitungen von $N_{i,p}(u)N_{j,q}(v)$. Auf den Knotenpunkten (u_i, v_j) (i, j wie oben) ist $N_{i,p}(u_i)N_{j,q}(v_j)$ in u-Richtung $(p-k_i)$-Mal differenzierbar und in v-Richtung, $(q-k_j)$-Mal differenzierbar. Dabei sind k_i und k_j die Ordnungen von u_i und v_j.

Weitere Eigenschaften sind deren der B-Spline-Kurven ähnlich.

vii. Die Oberfläche interpoliert die Endpunkte des Kontrollpolygons. Genauer gilt

$$S(a,c) = P_{0,0} \quad S(a,d) = P_{n,0} \quad S(b,c) = P_{0,m} \quad S(b,d) = P_{n,m}.$$

viii. B-Spline-Oberflächen sind invariant unter affinen Transformationen. Solche Transformationen werden nur auf das Kontrollpolygon angewendet.

ix. Die B-Spline-Oberfläche ist vollständig in der konvexen Hülle ihrer Kontrollpunkte enthalten.

x. Bewegt man einen Kontrollpunkt $P_{i,j}$ des Kontrollpolygons, so wirkt sich dies auf der B-Spline-Oberfläche nur auf dem Rechteck $[u_i, u_{i+p+1}) \times [v_j, v_{j+q+1})$ aus.

Eine Eigenschaft für B-Spline-Oberflächen, die der Eigenschaft vii. für B-Spline-Kurven entspricht, ist nicht bekannt. Den Beweis für diese Eigenschaften findet man ebenfalls in [10, Kapitel 1] oder in [8, Kapitel 3].

2.4 Partielle Ableitungen von B-Spline-Oberflächen

Entsprechend den B-Spline-Kurven werden Ableitungen von B-Spline-Oberflächen berechnet, indem man die B-Spline-Basisfunktionen ableitet. Der Unterschied ist, dass man hier an den partiellen Ableitungen der Ordnung d in den bidirektionalen Richtungen des Netzes der Kontrollpunkte interessiert ist.
Für fixes $(u, v) \in U \times V$ ist die allgemeine Formel gegeben durch

$$\frac{\partial^{k+l}}{\partial^k u\, \partial^l v} S(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} N_{i,p}^{(k)}(u) N_{j,q}^{(l)}(v) P_{i,j} \qquad 0 \leqslant k+l \leqslant d. \tag{2.11}$$

Benützt man also schrittweise die Methode zum Ableiten der B-Spline-Kurven, so entwickelt man die folgende Formel

$$S_{uv}(u,v) = \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} N_{i,p-1}(u) N_{j,q-1}(v) P_{i,j}^{(1,1)} \tag{2.12}$$

mit

$$P_{i,j}^{(1,1)} = q \frac{P_{i,j+1}^{(1,0)} - P_{i,j}^{(1,0)}}{v_{j+q+1} - v_{j+1}}. \tag{2.13}$$

Dabei werden von beiden Knotenvektoren U und V jeweils die Endpunkte weggenommen, und die Formel wird auf U' und V' berechnet.
Für höhere partielle Ableitungen kann man die folgende Formel

$$\frac{\partial^{k+l}}{\partial^k u\, \partial^l v} S(u,v) = \sum_{i=0}^{n-k} \sum_{j=0}^{m-l} N_{i,p-k}(u) N_{j,q-l}(v) P_{i,j}^{(k,l)} \quad 0 \leqslant k+l \leqslant d \tag{2.14}$$

mit

$$P_{i,j}^{(k,l)} = (q-l+1) \frac{P_{i,j+1}^{(k,l-1)} - P_{i,j}^{(k,l-1)}}{v_{j+q+1} - v_{j+1}} \tag{2.15}$$

herleiten, die auf den Knotenvektoren $U^{(k)}$ und $V^{(l)}$ berechnet wird.

2.5 Bézier Kurven

Einen ganz wichtigen Spezialfall der B-Splines bilden die Bézier Kurven, denn sie sind für verschiedene Anwendungen besser geeignet als B-Splines. Aus diesem

Grund sollen sie hier kurz erwähnt werden.

Definition 2.3 (Bézier Kurve)
Eine **Bézier Kurve** vom Grad p ist gegeben durch

$$C(u) = \sum_{i=0}^{n} B_i^p(u) P_i \qquad 0 \leqslant u \leqslant 1, \tag{2.16}$$

wobei P_i die Punkte des Kontrollpolygons und die $B_i^p(u)$, die klassischen **Bernsteinpolynome** vom Grad p sind, gegeben durch

$$B_i^p(u) = \binom{p}{i} u^i (1-u)^{p-i} = \frac{p!}{i!(p-i)!} u^i (1-u)^{p-i}. \tag{2.17}$$

Bernsteinpolynome sind Spezialfälle der B-Spline-Basisfunktionen für die Knotenvektoren der Form $U = \{\underbrace{0,\ldots,0}_{p+1}, \underbrace{1,\ldots,1}_{p+1}\}$ Die rekursive Darstellung ist gegeben durch folgende Gleichung

$$B_i^p(u) = (1-u) B_i^{p-1}(u) + u B_{i-1}^{p-1}(u),$$

wobei $B_i^p(u) = 0$ für $i < 0$ und $i > p$.
Außer den Eigenschaften aus Proposition 1.1 besitzen die Bernsteinpolynome noch zwei weitere interessante Eigenschaften:

- Jedes B_i^p hat genau ein Maximum auf dem Intervall $[0,1]$, und dieses liegt bei $u = \frac{i}{p}$.
- Für jedes p ist die Menge der Polynome $\{B_i^p(u)\}$ symmetrisch zu $\frac{1}{2}$.

Definition 2.4 (Bézier Oberflächen)
Bézier Oberflächen werden durch die Gleichung

$$S(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} B_i^p(u) B_j^q(v) P_{i,j}, \quad 0 \leqslant u,v \leqslant 1$$

definiert, wobei $P_{i,j}$ Kontrollpunkte des bidirektionalen Kontrollnetzes sind und $B_i^p B_j^q$ Tensorprodukte von Bernsteinpolynomen.

3 Rationale B-Spline-Kurven und Oberflächen - NURBS

Den zentralen Aspekt dieser Arbeit stellen nicht B-Spline-Kurven und B-Spline-Oberflächen dar, sondern eine spezielle Klasse von Splines, die sogenannten NURBS-Kurven und Oberflächen. In diesem Kapitel werden diese rationalen Funktionen vorgestellt, die aufgrund zusätzlicher Bauelemente, den Gewichten, eine weitere Möglichkeit bieten, ihren Verlauf zu modellieren.
Zusätzlich existiert eine Möglichkeit NURBS-Kurven als B-Spline-Kurven zu schreiben mithilfe der homogenen Koordinaten, und schließlich werden zum Schluss noch Formeln für Ableitungen von NURBS-Kurven angeführt.

3.1 Definitionen

Einige Kurven kann man nur mit rationalen Funktionen darstellen. In der Splines-Theorie verwendet man dazu rationale Splines, und NURBS (nonuniform rational B-Splines) sind ein Spezialfall davon. Sie werden auf Knotenvektoren von bestimmter Form definiert.

Definition 3.1 (NURBS-Kurven)
Eine **rationale B-Spline-Kurve vom Grad** p (eine NURBS-Kurve vom Grad p) ist gegeben durch

$$C(u) = \frac{\sum_{i=0}^{n} N_{i,p}(u) w_i P_i}{\sum_{j=0}^{n} N_{j,p}(u) w_j}, \quad 0 \leqslant u \leqslant 1, \tag{3.1}$$

wobei P_i für $i = 0, \ldots, n$ Kontrollpunkte des Kontrollpolygons, w_i die zuständigen Gewichte und $N_{i,p}$ die B-Spline-Basisfunktionen vom Grad p sind, definiert auf dem Knotenvektor

$$U = \{\underbrace{0, \ldots, 0}_{p+1}, u_{p+1}, \ldots, u_{m-p-1}, \underbrace{1, \ldots, 1}_{p+1}\}.$$

Um die Schreibweise abzukürzen, setzt man

$$R_{i,p}(u) = \frac{N_{i,p}(u)w_i}{\sum_{j=0}^{n} N_{j,p}(u)w_j}. \tag{3.2}$$

Die $R_{i,p}$ für $i = 0, \ldots, n$ nennt man **rationale Basisfunktionen**.

Die NURBS-Kurve wird dementsprechend durch die Gleichung

$$C(u) = \sum_{i=0}^{n} R_{i,p}(u)P_i \tag{3.3}$$

dargestellt.

Die rationalen Basisfunktionen $R_{i,p}$ besitzen dieselben Eigenschaften (*i.-v.* aus Proposition 1.1), wie die B-Spline-Basisfunktionen. Ist $w_i = a$ für $a \neq 0$ für alle $i = 0, \ldots, n$, dann gilt $R_{i,p} = N_{i,p}$. Somit sind die B-Spline-Basisfunktionen ein Spezialfall der rationalen Basisfunktionen.
NURBS-Kurven haben die Eigenschaften *ii.*, *iii.*, *iv.* und *vii.* aus Proposition 2.1, die für B-Spline-Kurven gelten. Die Eigenschaft *v.* aus Proposition 2.1 ist ebenfalls gültig. Sie bezieht sich aber nicht nur auf die Kontrollpunkte. Wird ein Kontrollpunkt P_i oder das zugehörige Gewicht w_i verändert, so hat dies Auswirkungen auf die NURBS-Kurve $C(u)$ nur im Intervall $[u_i, u_{i+p+1})$.

Die Variation der Gewichte stellt nämlich eine weitere Möglichkeit dar, die Form der Kurve zu ändern. Je höher ein Gewicht w_i ist, desto näher ist die NURBS-Kurve dem zugehörigen Kontrollpunkt P_i, wie das folgende Bild verdeutlicht.

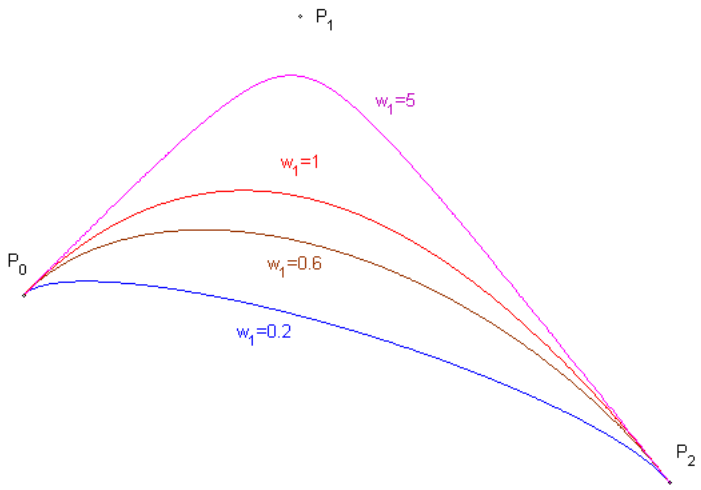


Abbildung 3.1: variable Gewichte

Die NURBS-Kurve ist vom Grad $p = 2$ mit den Kontrollpunkten $P_0 = (1.5,\ 5)$, $P_1 = (5.5,\ 5)$, und $P_2 = (9.5,\ 0)$. Die Gewichte w_0 und w_2 sind beide jeweils 1, nur das mittlere Gewicht w_1 wird variiert.

Definition 3.2 (NURBS-Oberflächen)
Eine **NURBS-Oberfläche vom Grad** p in Richtung u und vom Grad q in Richtung v ist eine bivariante stückweise rationale Funktion, gegeben durch die Gleichung

$$S(u,v) = \frac{\sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u), N_{j,q}(v) w_{i,j} P_{i,j}}{\sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u), N_{j,q}(v) w_{i,j}}, \quad 0 \leqslant u,v, \leqslant 1. \tag{3.4}$$

Für $i = 0,\dots,n$ und $j = 0,\dots,m$ bilden die Kontrollpunkte $P_{i,j}$ ein bidirektionales Kontrollnetz, $w_{i,j}$ sind die zugehörigen Gewichte, $N_{i,p}$ und $N_{j,q}$ die B-Spline-Basisfunktionen jeweils definiert auf den Knotenvektoren U und V der Form

$$\begin{aligned} U =& \{\underbrace{0,\dots,0}_{p+1}, u_{p+1}\dots,u_{r-p-1}, \underbrace{1,\dots,1}_{p+1}\} \\ V =& \{\underbrace{0,\dots,0}_{q+1}, u_{q+1}\dots,u_{s-q-1}, \underbrace{1,\dots,1}_{q+1}\}, \end{aligned}$$

wobei $r = n + p + 1$ und $s = m + q + 1$.
Die rationalen Basisfunktionen sind somit gegeben durch die Gleichung

$$R_{i,j}(u,v) = \frac{N_{i,p}(u), N_{j,q}(v) w_{i,j}}{\sum_{k=0}^{n}\sum_{l=0}^{m} N_{k,p}(u), N_{l,q}(v) w_{k,l}}, \tag{3.5}$$

wodurch die NURBS-Oberfläche die Gestalt

$$S(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} R_{i,j}(u,v) P_{i,j} \tag{3.6}$$

annimmt.

Die NURBS-Oberflächen haben einige Eigenschaften, die auch für B-Spline-Oberflächen gelten (Proposition 2.2. *i.-ix.*). Der Punkt *x.* kann auch auf die Gewichte $w_{i,j}$ erweitert werden: Die Änderungen der Kontrollpunkte $P_{i,j}$ oder der zugehörigen Gewichte $w_{i,j}$ beeinflussen die NURBS-Oberfläche nur auf dem Rechteck $[u_i, u_{i+p+1}) \times [v_j, v_{j+q+1})$. Die Änderung der Gewichte ist ebenfalls eine Möglichkeit, die Form der Oberfläche lokal zu ändern.

3.2 Homogene Koordinaten

Da NURBS-Kurven und Oberflächen im Prinzip nur einen Nenner haben, empfiehlt es sich, bei der Implementierung von Algorithmen homogene Koordinaten zur verwenden. Dadurch kann man rationale Kurven im $\mathbb{R}^n$ als Polynomkurven im $\mathbb{R}^{n+1}$ darstellen.
Für einen Punkt $T = (x, y, z) \in \mathbb{R}^3$ sei $T^w = (wx, wy, wz, w)$ mit $w \neq 0$ die homogene Darstellung in $\mathbb{R}^4$.
Diese Beziehung wird durch die perspektive Abbildung $H : \mathbb{R}^4 \to \mathbb{R}^3$

$$H(T^w) = H(wx, wy, wz, w) = (\frac{wx}{w}, \frac{wy}{w}, \frac{wz}{w}) = (x, y, z) \tag{3.7}$$

definiert.
Die Darstellung ist dabei unabhängig von der Wahl von w, denn für $w_1 \neq w_2$ ist

$$H(T^{w_1}) = H(w_1x, w_1y, w_1z, w_1) = (x, y, z) = H(w_2x, w_2y, w_2z, w_2) = H(T^{w_2}).$$

Um NURBS-Kurven durch homogene Koordinaten darzustellen, bildet man zuerst aus den Kontrollpunkten $P_i = (x_i, y_i, z_i)$ und den zugehörigen Gewichten w_i gewichtete Kontrollpunkte $P_i^w = (w_ix_i, w_iy_i, w_iz_i, w_i)$. Dann definiert man die stückweise polynomiale Kurve,

$$C^w(u) = \sum_{i=0}^{n} N_{i,p} P_i^w$$

im höherdimensionalen Raum.
Man wendet die perspektive Abbildung H aus Gleichung (3.7) auf $C^w(u)$ an, um die NURBS-Kurve zurück zu bekommen:

$$\begin{aligned} C(u) = H(C^w(u)) &= H\left(\sum_{i=0}^{n} N_{i,p} P_i^w\right) \\ &= \frac{\sum_{i=0}^{n} N_{i,p}(u) w_i P_i}{\sum_{i=0}^{n} N_{i,p}(u) w_i}. \end{aligned}$$

Besitzen alle Gewichte den Wert 1, so erhält eine B-Spline-Kurve, und damit sind B-Spline-Kurven Spezialfälle von NURBS-Kurven.

Dasselbe Konzept kann auf NURBS-Oberflächen angewendet werden:

$$S^w(u, v) = \sum_{i=0}^{n} \sum_{j=0}^{m} N_{i,p}(u) N_{j,q}(v) P_{i,j}^w,$$

wobei $P^w_{i,j} = (w_{i,j}x_{i,j}, w_{i,j}y_{i,j}, w_{i,j}z_{i,j}, w_{i,j})$. Den Punkt $C = S(u,v)$ berechnet man, indem man die perspektive Abbildung H auf $S^w(u,v)$ anwendet, also $H(S^w(u,v))$ berechnet.
In diesem Sinne erhält man für $S^w(u,v)$ eine Darstellung der Oberfläche als Tensorprodukt, jedoch gilt dies nicht für $S(u,v)$, da es nicht als Produkt von Basisfunktionen gegeben ist.

3.3 Ableitungen von NURBS-Kurven

Obwohl man NURBS-Kurven in homogenen Koordinaten in der B-Splines-Form aufschreiben kann, darf man sie so nicht mithilfe der Formeln (2.4) und (2.5) ableiten, da die Gewichte w_i auch abgeleitet werden müssen.
Die erste Ableitung der NURBS-Kurve $C(u) = \frac{\sum_{i=0}^n N_{i,p}(u)P_i w_i}{\sum_{j=0}^n N_{j,p}(u)w_j}$ ist gegeben durch

$$C'(u) = \frac{\sum_{i=0}^n N'_{i,p}(u)P_i w_i \cdot \sum_{j=0}^n N_{j,p}(u)w_j - \sum_{i=0}^n N_{i,p}(u)P_i w_i \cdot \sum_{j=0}^n N'_{j,p}(u)w_j}{\left(\sum_{j=0}^n N_{j,p}(u)w_j\right)^2} \tag{3.8}$$

$$= \frac{\sum_{i=0}^n N'_{i,p}(u)P_i w_i}{\sum_{j=0}^n N_{j,p}(u)w_j} - C(u)\frac{\sum_{j=0}^n N'_{j,p}(u)w_j}{\sum_{j=0}^n N_{j,p}(u)w_j}. \tag{3.9}$$

Etwas komplizierter sieht die zweite Ableitung aus:

$$\begin{aligned}
C''(u) = {} & \frac{\sum_{i=0}^n N''_{i,p}(u)P_i w_i \cdot \sum_{j=0}^n N_{j,p}(u)w_j - \sum_{i=0}^n N'_{i,p}(u)P_i w_i \cdot \sum_{j=0}^n N'_{j,p}(u)w_j}{\left(\sum_{j=0}^n N_{j,p}(u)w_j\right)^2} \\
& - C'(u)\frac{\sum_{j=0}^n N'_{j,p}(u)w_j}{\sum_{j=0}^n N_{j,p}(u)w_j} \\
& - C(u)\frac{\sum_{j=0}^n N''_{j,p}(u)w_j \cdot \sum_{j=0}^n N_{j,p}(u)w_j - \sum_{j=0}^n N'_{j,p}(u)w_j \cdot \sum_{j=0}^n N'_{j,p}(u)w_j}{\left(\sum_{j=0}^n N_{j,p}(u)w_j\right)^2} \\
= {} & \frac{\sum_{i=0}^n N''_{i,p}(u)P_i w_i}{\sum_{j=0}^n N_{j,p}(u)w_j} - \frac{\sum_{i=0}^n N'_{i,p}(u)P_i w_i \cdot \sum_{j=0}^n N'_{j,p}(u)w_j}{\left(\sum_{j=0}^n N_{j,p}(u)w_j\right)^2} - C'(u)\frac{\sum_{j=0}^n N'_{j,p}(u)w_j}{\sum_{j=0}^n N_{j,p}(u)w_j} - \\
& - C(u)\frac{\sum_{j=0}^n N''_{j,p}(u)w_j}{\sum_{j=0}^n N_{j,p}(u)w_j} + C(u)\frac{\left(\sum_{j=0}^n N'_{j,p}(u)w_j\right)^2}{\left(\sum_{j=0}^n N_{j,p}(u)w_j\right)^2}.
\end{aligned}$$

Setzt man weiters $A(u) = \sum_{i=0}^n N_{i,p}(u)P_i w_i$ und $g(u) = \sum_{j=0}^n N_{j,p}(u)w_j$, so ist $C(u) = \frac{A(u)}{g(u)}$, und für Ableitungen höheren Grades ergibt sich die Formel

$$C^{(k)}(u) = \frac{1}{g(u)}\left(A^{(k)}(u) - \sum_{i=1}^k \binom{k}{i} g^{(i)}(u)C^{(k-i)}(u)\right). \tag{3.10}$$

4 Die wichtigsten B-Spline-Algorithmen

In diesem Kapitel werden kurz, meistens ohne auf die Details genauer einzugehen, und auch ohne Implementierungen, die wichtigsten Algorithmen für B-Splines vorgestellt. Die wenigen Implementierungen, die doch angegeben werden, sind im entsprechendem Kapitel zu finden.
Zum Schluss wird noch ein Algorithmus angegeben, der B-Spline-Kurven in Bézier-Segmente zerlegt.

4.1 Auswertungsalgorithmen, „De Boor Algorithmus“

Nun da man NURBS-Kurven definiert hat, möchte man sie auch auswerten. Im folgenden werden zwei Algorithmen vorgestellt, die dies durchführen.

4.1.1 Matrixmultiplikation

Das einfachste Verfahren, um eine NURBS-Kurve $C(u) = \frac{\sum_{i=0}^{n} N_{i,p}(u) w_i P_i}{\sum_{j=0}^{n} N_{j,p}(u) w_j}$ auszuwerten, ist die Matrixmultiplikation.

Dabei werden für einen Punkt $u \in [u_0, u_m]$ die Werte der Basisfunktionen an dieser Stelle berechnet. Sei

$$A = \left(N_{0,p}(u), N_{1,p}(u), \ldots, N_{n,p}(u)\right)$$

der Zeilenvektor, der diese Werte beinhaltet. Als nächstes bildet man mithilfe der perspektiven Abbildung H aus (3.7) die homogenen Koordinaten der Kontrollpunkte und fasst sie im Spaltenvektor zusammen.

$$P = \left(P_0^w, P_1^w, \ldots, P_n^w\right)^t.$$

Multipliziert man diese beiden Vektoren miteinander, so erhält man den Wert $C^w(u)$, also den Wert der NURBS-Kurve in homogenen Koordinaten an der Stelle u. Schließlich muss man noch die Inverse der perspektiven Abbildung H auf $C^w(u)$ anwenden, um den Wert $C(u)$ zu bekommen.

Aufgrund der Eigenschaft *ii.* aus Proposition 1.1 kann man den Aufwand für diese Vorgangsweise auch verringern.
Zuerst muss man bestimmen, in welchem Teilintervall $[u_i, u_i + 1)$ des Knotenvektors U sich der Wert u befindet. Sei j der gesuchte Index. Da auf $[u_j, u_j + 1)$ nur die $p + 1$ Basisfunktionen $N_{j-p,p}, \ldots, N_{j,p}$ ungleich Null sind, gilt dies ebenfalls für den Wert u. Aus diesem Grund kann man den Vektor A auf den Vektor $\tilde{A} = \left(N_{j-p,p}(u), N_{j-p-1,p}(u), \ldots, N_{j,p}(u)\right)$ reduzieren.
Dieselbe Reduktion wird auch auf den Vektor P vorgenommen, und man erhält den Vektor $\tilde{P} = \left(P^w_{j-p}, \ldots, P^w_j\right)^t$. Schließlich bekommt man einen Algorithmus, der zwei Vektoren der Länge $p + 1$ multipliziert.

Der Algorithmus 'CurvePoint2', dessen Implementation im letzten Kapitel zu finden ist, leistet genau das oben Beschriebene. Die Vorlage für die Implementierung lieferte [8, Kapitel 2 und 4].

Für NURBS-Oberflächen in homogenen Koordinaten, mit der Formel

$$S^w(u, v) = \sum_{i=0}^{n} \sum_{j=0}^{m} N_{i,p}(u) N_{j,q}(v) P^w_{i,j},$$

ist der Algorithmus etwas komplexer, da die homogenen Kontrollpunkte in einer Matrix angeordnet werden müssen. Zusätzlich zum Vektor A, der $N_{i,p}(u)$ enthält, muss man noch den Vektor B berechnen, der die $N_{j,q}(v)$ enthält.
Den Wert $S^w(u, v)$ erhält man, indem man die Matrixmultiplikation APB^t durchführt.
Die oben beschriebenen Reduktionsschritte auf Teilvektoren kann man hier auch durchführen.

4.1.2 De Boor Algorithmus

Ein etwas effizienterer, rekursiver Algorithmus zum Auswerten der Spline-Kurven wurde von Carl. R. de Boor entwickelt. Dieser wird hier vorgestellt.

Sei die B-Spline-Kurve $C(u) = \sum_{i=0}^{n} N_{ip}(u) P_i$ auf dem Knotenvektor $U = \{u_0, \ldots, u_m\}$ definiert, wobei alle Knoten in U die Ordnung $\leqslant p$ haben. Sei weiters $u \in [u_j, u_{j+1})$ für ein festes j. Um die Schreibweise abzukürzen, setzt man

$$r_{i,k}(u) = \begin{cases} \frac{u - u_i}{u_{i+k} - u_i}, & \text{für } u_{i+k} \neq u_i \\ 0 & \text{sonst} \end{cases} \qquad \text{für } i = 0, \ldots, m - p - 1 \text{ und } k = 0, \ldots, p.$$

Dann kann man die B-Spline-Basisfunktionen schreiben als

$$N_{i,p}(u) = r_{i,p}(u)N_{i,p-1}(u) + (1 - r_{i+1,p}(u))N_{i+1,p-1}(u).$$

Der de Boor-Algorithmus basiert darauf, dass man die Spline-Kurve auf folgende Art umschreiben kann:

$$C(u) = \sum_{i=0}^{n} N_{i,p}(u)P_i = \sum_{i=j-p}^{j} N_{i,p}(u)P_i^{(0)} = \sum_{i=j-p+1}^{j} N_{i,p-1}(u)P_i^{(1)} \tag{4.1}$$

$$= \sum_{i=j-p+2}^{j} N_{i,p-2}(u)P_i^{(2)} = \ldots = \sum_{i=j}^{j} N_{i,p-p}(u)P_i^{(p)} = P_i^{(p)}, \tag{4.2}$$

wobei

$$P_i^{(0)} = P_i \tag{4.3}$$

$$P_i^{(t+1)} = r_{i,p-t}(u)P_i^t + (1 - r_{i,p-t}(u))P_{i-1}^t. \tag{4.4}$$

Im Folgenden ist angeführt, wieso (4.1) gilt. Die restlichen Identitäten aus der obigen Gleichung, berechnet man analog.
Es ist nämlich

$$\begin{aligned} C(u) &= \sum_{i=j-p}^{j} N_{i,p}(u)P_i = \sum_{i=j-p}^{j} \left(r_{i,p}(u)N_{i,p-1}(u) + (1 - r_{i+1,p}(u))N_{i+1,p-1}(u)\right) P_i = \\ &= \sum_{i=j-p}^{j} r_{i,p}(u)N_{i,p-1}(u)P_i + \sum_{i=j-p+1}^{j+1} (1 - r_{i,p}(u))N_{i,p-1}(u)P_{i-1} = \\ &= r_{j-p,p}(u)N_{j-p,p-1}(u)P_{j-p} + \sum_{i=j-p+1}^{j} \left(r_{i,p}(u)P_i + (1 - r_{i,p}(u)P_{i-1})\right) N_{i,p-1}(u) \\ &\quad + (1 - r_{j+1,p}(u))N_{j+1,p-1}(u)P_j. \end{aligned}$$

Wegen der Eigenschaft *ii.* aus Proposition 1.1, sind $N_{j-p,p-1}(u) = 0$ und $N_{j+1,p-1}(u) = 0$, womit die Identität gezeigt wird.

Das Praktische an dem Verfahren ist, dass man die P_i^t in einem Tabelau aufreihen kann:

$$
\begin{array}{ccccccccccccc}
P_{j-p}^{(0)} & & P_{j-p+1}^{(0)} & & P_{j-p+2}^{(0)} & \cdots & & & & & \cdots & & P_{j}^{(0)} \\
& P_{j-p+1}^{(1)} & & P_{j-p+2}^{(1)} & \cdots & \cdots & & & & \cdots & P_{j}^{(1)} & & \\
& & P_{j-p+2}^{(2)} & & P_{j-p+3}^{(2)} & \cdots & & & \cdots & P_{j}^{(2)} & & & \\
& & & P_{j-p+3}^{(3)} & & \cdots & & & P_{j}^{(3)} & & & & \\
& & & & \ddots & & & & & & & & \\
& & & & & & & \cdots & & & & & \\
& & & & & & P_{j-1}^{(p-1)} & & P_{j}^{(p-1)} & & & & \\
& & & & & & & P_{j}^{(p)} & & & & &
\end{array}
$$

Jeder Punkt ist eine lineare Konvexkombination aus den Punkten oberhalb.

Geometrisch berechnet der De Boor Algorithmus in jedem Schritt neue Kontrollpunkte, sodass sich das Kontrollpolygon dem gesuchten Punkt der Kurve nähert. Der letzte berechnete Punkt $P_j^{(p)}$ liegt schließlich auf der Kurve. Der Vorgang vermittelt den Eindruck, dass die Ecken des Kontrollpolygons in jedem Schritt abgeschnitten werden.

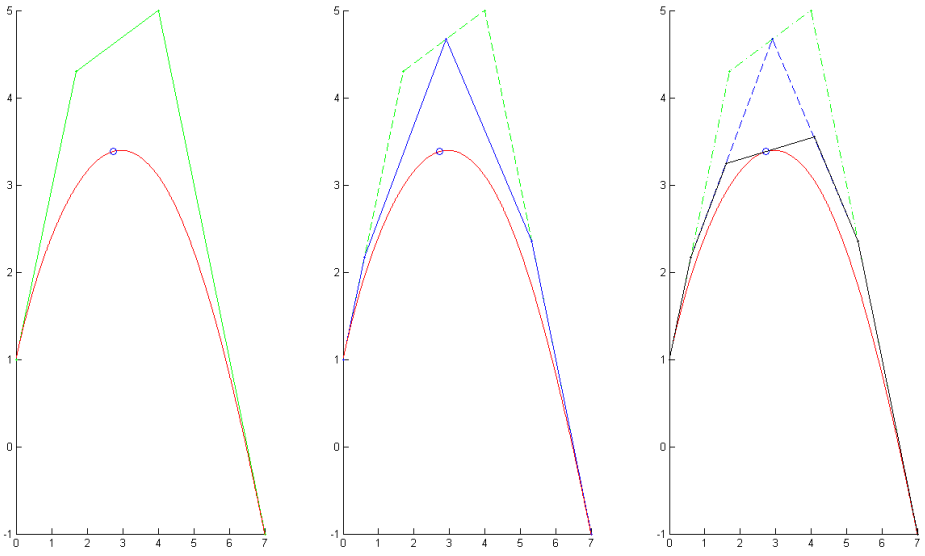


Abbildung 4.1: De Boor Algorithmus

Den De Boor Algorithmus kann man auf NURBS-Kurven in homogenen Koordinaten anwenden. Genau dies leistet der Algorithmus 'CurvePntByCornerCut', dessen Implementation auch im letzten Kapitel angeführt ist. Die Vorlage dazu liefert erneut [8, Kapitel 5].

4.2 Knoteneinfügung - Knot Insertion

Wie der Name besagt, ist „Knot insertion“ ein Algorithmus, der dazu dient einen zusätzlichen Knoten $\hat{u}$ dem Knotenvektor U hinzuzufügen.
Sei $C^w(u) = \sum_{i=0}^{n} N_{i,p}(u)$ eine NURBS-Kurve, die auf $U = \{u_0, \ldots, u_m\}$ definiert ist. Fügt man ein $\hat{u}$ dem Knotenintervall $[u_j, u_{j+1})$ hinzu, wobei $u_j \leqslant \hat{u} \leqslant u_{j+1}$ für ein j, so ergibt sich ein neuer Knotenvektor der Form

$$\hat{U} = \{\underbrace{\hat{u}_0}_{=u_0}, \underbrace{\hat{u}_1}_{=u_1}, \ldots, \underbrace{\hat{u}_j}_{=u_j}, \underbrace{\hat{u}_{j+1}}_{\hat{u}}, \underbrace{\hat{u}_{j+2}}_{u_{j+1}}, \ldots, \underbrace{\hat{u}_{m+1}}_{u_m}\}.$$

Dieselbe NURBS-Kurve $C^w(u)$ kann auch auf $\hat{U}$ definiert werden durch

$$C^w(u) = \sum_{i=0}^{n+1} \hat{N}_{i,p}(u) Q_i^w, \tag{4.5}$$

wobei $\hat{N}_{i,p}$ B-Spline-Basisfunktionen vom Grad p sind, definiert auf $\hat{U}$.
Die Q_i^w sind Punkte des neuen Kontrollpolygons. Sie werden in den Iterationen des „Knot insertion“ Algorithmus neu berechnet.
Der Algorithmus verändert die B-Spline-Kurve weder geometrisch (Länge, Krümmung, Form,...) noch den Parametervektor der Kurve. Um die Q_i^w zu berechnen, muss man das Gleichungssystem

$$\sum_{i=0}^{n} N_{i,p}(u) P_i^w = \sum_{i=0}^{n+1} \hat{N}_{i,p}(u) Q_i^w \tag{4.6}$$

aus $n+2$ linearen Gleichungen lösen. Aufgrund der Eigenschaft *ii.* aus Proposition 1.1 ist es ausreichend, das Gleichungssystem

$$\sum_{i=k-p}^{k} N_{i,p}(u) P_i^w = \sum_{i=k-p}^{k+1} \hat{N}_{i,p}(u) Q_i^w \tag{4.7}$$

aus $p+2$ Gleichungen für $u \in [u_k, u_{k+1})$ zu lösen.
Für Basisfunktionen und Kontrollpunkte außerhalb dieses Teilintervalls gilt:

$$N_{i,p}(u) = \hat{N}_{i,p}(u) \qquad P_i^w = Q_i^w \qquad i = 0, \ldots, k-p-1 \tag{4.8}$$

$$N_{i,p}(u) = \hat{N}_{i+1,p}(u) \qquad P_i^w = Q_{i+1}^w \qquad i = k+1, \ldots, n. \tag{4.9}$$

Um die Gleichung (4.7) zu lösen, drückt man zuerst die $N_{i,p}$ durch $\hat{N}_{i,p}$ aus:

$$N_{i,p}(u) = \frac{\hat{u} - \hat{u}_i}{\hat{u}_{i+p+1} - \hat{u}_i} \hat{N}_{i,p}(u) + \frac{\hat{u}_{i+p+2} - \hat{u}}{\hat{u}_{i+p+2} - \hat{u}_{i+1}} \hat{N}_{i+1,p}(u). \tag{4.10}$$

Dies setzt man in die Gleichung (4.7) ein und führt einen Koeffizientenvergleich durch. Nach weiteren Umformungen ergibt sich der Algorithmus zur Berechnung der Q_i^w:

$$Q_i^w = P_i^w \qquad 0 \leqslant i \leqslant k-p-1 \tag{4.11}$$

$$Q_i^w = \alpha_i P_i^w + (1-\alpha_i) P_{i-1}^w \qquad k-p+1 \leqslant i \leqslant k \tag{4.12}$$

$$Q_{i+1}^w = P_i^w \qquad k+1 \leqslant i \leqslant n, \tag{4.13}$$

wobei

$$\alpha_i = \begin{cases} 1 & 1 \leqslant k-p \\ \frac{\hat{u}-u_i}{u_{i+p}-u_i} & k-p+1 \leqslant i \leqslant k \\ 0 & i \geqslant k+1. \end{cases} \tag{4.14}$$

Es müssen also nur p neue Punkte berechnet werden.

BEISPIEL
Hier sieht man die B-Spline-Kurve vom Grad $p = 3$, die auf dem Knotenvektor $U = \{0,0,0,0,0.5,1,1,1,1\}$ definiert ist, mit den Kontrollpunkten $P_0 = (1,\ 1)$, $P_1 = (2.5,\ 4)$, $P_2 = (4.5,\ 4)$, $P_3 = (5.5,\ 1)$ und $P_4 = (7.5,\ 1)$. Das zugehörige Kontrollpolygon ist schwarz dargestellt.
Nach dem Hinzufügen von $u = 0.1$ zum Knotenvektor U hat sich die Kurve selbst nicht verändert, doch das Kontrollpolygon schon. Es ist ein weiterer Punkt hinzugekommen, sodass das neue Kontrollpolygon aus den Punkten $Q_0 = (1,\ 1)$, $Q_1 = (1.25,\ 1.5)$, $Q_2 = (2.7,\ 4)$, $Q_3 = (4.622,\ 3.6341)$, $Q_4 = (5.5,\ 1)$ und $Q_5 = (7.5,\ 1)$ besteht.

Um den Knoten $\hat{u}$ öfter als einmal dem Knotenvektor U hinzuzufügen, entwickelt man den oberen Algorithmus weiter.
Ist $\hat{u}$ schon in U und hat dort die Ordnung s und fügt man $\hat{u}$ weitere r-mal hinzu ($r+s \leqslant p$), so sei $Q_{i,r}^w$ der Punkt, der beim letzten Hinzufügen entsteht. ($Q_{i,0}^w = Q_i^w$). Er wird berechnet durch

$$Q_{i,r}^w = \alpha_{i,r} Q_{i,r-1}^w (1-\alpha_{i,r}) Q_{i-1,r-1}^w \tag{4.15}$$

mit

$$\alpha_{i,r} = \begin{cases} 1 & 1 \leqslant k-p+r-1 \\ \frac{\hat{u}-u_i}{u_{i+p-r+1}-u_i} & k-p+r \leqslant i \leqslant k-s \\ 0 & i \geqslant k-s+1. \end{cases} \tag{4.16}$$

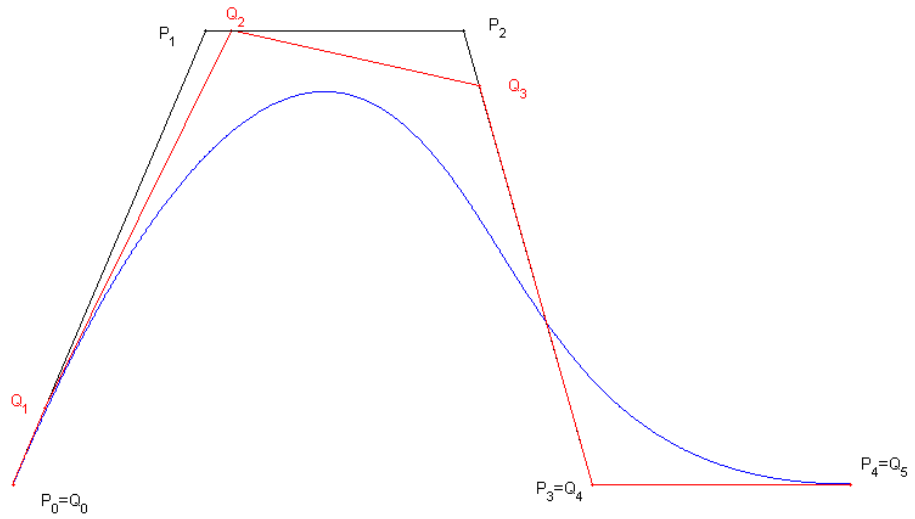


Abbildung 4.2: Knot Insert

Durch das Hinzufügen eines jeden neuen Knotens entstehen neue Kontrollpunkte. Damit werden interaktive Änderungen des Verlaufs der Kurve um einiges einfacher.

Als einer der wohl wichtigsten Algorithmen der Spline-Theorie besitzt „Knot insertion“ einige Anwendungen. Eine davon ist das Unterteilen einer Kurve.

Unterteilen der Kurve

Fügt man einen Knoten $\hat{u}$ in $[u_j, u_{j+1})$ (für ein j) so oft ein, dass seine Ordnung größer als $p+1$ wird, so wird die Kurve C in diesem Punkt $P = C(\hat{u})$ in zwei Kurven C_l und C_r unterteilt. Die zwei Kurvenstücke kann man unabhängig voneinander als selbstständige B-Spline-Kurven vom selben Grad betrachten, die jeweils auf $U_l = \{\underbrace{u_0}_{p+1}, u_1, \ldots, u_j, \underbrace{\hat{u}}_{p+1}\}$ und auf $U_r = \{\underbrace{\hat{u}}_{p+1}, u_{j+1}, \ldots, \underbrace{u_m}_{p+1}\}$ definiert sind.

Berechnen der Punkte auf der Kurve

Sei $Q^w_{i,r}$ der Kontrollpunkt, der entsteht, wenn man einen Knoten $\hat{u}$ r-mal in U einfügt. Dieser neu entstandene Kontrollpunkt liegt näher an der Kurve als die ursprünglichen Kontrollpunkte. Fügt man $\hat{u}$ oft genug ein, so liegt einer der neu

entstandenen Punkte auf der Kurve selbst.
Das ist eine weitere Möglichkeit Punkte der Kurve zu berechnen, ohne den rekursiven Algorithmus zu benützen. Diese Methode wird oft zur Darstellung der Kurven am Bildschirm verwendet, anstatt einen Auswertungsalgorithmus zu benützen.
Vergleicht man nämlich den De Boor Algorithmus (4.3) mit dem Knot Insertion Verfahren (4.14), so sind nur minimale Unterschiede zu erkennen.

4.2.1 Umgekehrte Knoteneinfügung

Arbeitet man interaktiv mit NURBS-Kurven, so weiß man intuitiv, welchen Punkt des Kontrollpolygons man ändern will, oder wo man einen neuen erzeugen möchte. Weiß man jedoch nicht, welchen Knoten man dem Knotenvektor hinzufügen muss, damit der neue Punkt entsteht, so wendet man den Umgekehrten Knot Insertion Algorithmus an.

Sei Q^w ein Punkt auf dem Kontrollpolygon, und sei $Q^w \neq P_i$ für alle $i = 1, \dots, n$, dann liegt Q^w auf der Strecke zwischen P_i^w und P_{i+1}^w für ein fixes i. Daher existiert ein $0 \leqslant s \leqslant 1$ so, dass

$$Q^w = (1-s)P_{i-1}^w + sP_i$$

und

$$Q = \frac{(1-s)w_{i-1}P_{i-1} + sw_iP_i}{(1-s)w_{i-1} + sw_i}.$$

Dann ist

$$s = \frac{w_{i-1}|Q - P_{i-1}|}{w_{i-1}|Q - P_{i-1}| + w_i|P_i - Q|},$$

und somit ist der Knoten $\hat{u}$, den man einfügen muss, um Q zu erzeugen, gegeben durch

$$\hat{u} = u_i + s(u_{i+p} - u_i).$$

Dieser Algorithmus funktioniert für NURBS-Oberflächen folgendermaßen: Um $(\hat{u}, \hat{v})$ zu berechnen, muss $\hat{u}$ in alle $m+1$ Reihen des bidirektionalen Netzes, das $U \times V$ bildet, eingefügt werden und analog $\hat{v}$ in alle $n+1$ Spalten.

4.2.2 Entfernen von Knoten

Man kann Knoten aus einem Knotenvektor auch entfernen. Diese Vorgehensweise wird verwendet, um Speicherplatz zu sparen, bzw. die kompakteste Form der Spline-Kurve zu erhalten.
Sei $C^w(u) = \sum_{i=0}^{n} N_{i,p}(u)P_i^w$ eine NURBS-Kurve, die auf $U = \{u_0, \ldots, u_m\}$ definiert ist. Sei weiters u_j ein Knoten in U mit Ordnung $t > 1$, und sei U_1 der Knotenvektor, der entsteht, wenn man u_r einmal aus U entfernt. Dabei heißt u_r „entfernbar“, wenn $C^w(u)$ eine Darstellung der Form

$$C^w(u) = \sum_{i=0}^{n-1} \tilde{N}_{i,p}(u)Q_i^w$$

auf U_1 besitzt.
Die Punkte Q_i^w berechnet man wie folgt:

$$\begin{aligned}
Q_i^w &= P_i^w \qquad \text{für } i = 0, \ldots, j-p-1 \\
Q_i^w &= \frac{P_i^w - (1-\alpha_i)Q_{i-1}^w}{\alpha_i} \qquad \text{für } i = j-p, \ldots, \frac{1}{2}(2j-p-t-1) \\
Q_i^w &= \frac{P_i^w - \alpha_i Q_{i+1}^w}{1-\alpha_i} \qquad \text{für } i = \frac{1}{2}(2j-p-t+2), \ldots, j-t \\
Q_i^w &= P_i^w \qquad \text{für } i = j-t+1, \ldots, n-1
\end{aligned}$$

mit

$$\alpha_k = \frac{u - u_k}{u_{k+p+1} - u_k}.$$

Durch das Entfernen eines mehrfachen Knotenpunktes sollen weder geometrische Eigenschaften noch die Parametermenge der Kurve verändert werden.
Die Fragen, die sich hier aufdrängen, sind: Ist der Knoten überhaupt entfernbar? Wenn ja, "wie oft"?
Für die NURBS-Kurve $C^w(u)$ in homogenen Koordinaten wird erwartet, dass sie auf dem Knoten u_j mindestens C^{p-t} mal differenzierbar ist. Die Kontrollpunkte können aber so liegen, dass die Kurve auf u_r öfter differenzierbar ist als $p-t$-mal. Deswegen ist der Knoten u_r mindestens t-mal entfernbar, wenn C^w auf u_r mindestens C^{p-t+s} ist.
Details zu diesem Algorithmus findet man in [13].

Zerlegen einer B-Spline-Kurve in Bézier-Segmente

In manchen Anwendungen ist es praktischer, statt B-Spline-Kurven Bézier Kurven zu verwenden. In diesem Unterabschnitt wird kurz ein Algorithmus angedeutet, der dies leistet.

Sei $C(u) = \sum_{i=0}^{n} N_{i,p}(u)P_i$ eine B-Spline-Kurve definiert auf

$$U = \{u_0, \ldots, u_m\} = \{\underbrace{0, \ldots, 0}_{p+1}, \underbrace{u_1, \ldots, u_1}_{m_1}, \ldots, \underbrace{u_s, \ldots, u_s}_{m_s}, \underbrace{1, \ldots, 1}_{p+1}\}.$$

Um diese Kurve in ihre Polynomsegmente zu zerlegen, muss man jeden Knoten u_i ($i = 1, \ldots, s$) dem Knotenvektor U so oft hinzufügen, bis die Ordnungen aller Knoten jeweils $p + 1$ betragen. Auf diese Art kann die Kurve C in Teilkurven, die eigentlich Bézier Kurven entsprechen, zerlegt werden.
Der Algorithmus besteht aus vier Hauptschritten:

i. Finde Indizes a und b, sodass $u_a \leqslant z \leqslant u_b$ für alle $z \in Z$.

ii. Die Kontrollpunkte $P_0, \ldots, P_{a-p}$ und $P_{b-p}, \ldots, P_n$ werden nicht verändert. Diese sollen gespeichert und am Anfang sowie am Ende des Arrays mit den neu berechneten Kontrollpunkten hinzugefügt werden. (Dazwischen liegen noch $r + p + b - a - 1$ weitere neue Kontrollpunkte, die man berechnen muss.)

iii. Erzeuge $\hat{U} = U \cup Z$ mit $\hat{u}_i \leqslant \hat{u}_{i+1}$ für alle $u_i \in \hat{U}$.

iv. Berechne die fehlenden Kontrollpunkte.

Der letzte Schnitt, kann mit dem Knot Insertion Algorithmus erreicht werden. Da die meisten Knoten mehr als einmal eingefügt werden müssen, verwendet man die Formeln (4.15) und (4.16).
Dabei entsteht für die $\alpha_{i,r}$ wieder ein trianguläres Schema, wobei sich die Werte diagonal wiederholen.
Diese Sachverhalte werden detaillierter in [8, Kapitel 5] angeführt.

5 Intervallarithmetik

Da die Implementation im letzten Kapitel mithilfe von Intervallen durchgeführt werden, gibt dieses Kapitel eine kurze Einführung in die Intervallarithmetik. Auch das Intervall-Newton-Verfahren wird erklärt.

5.1 Rechnen mit Intervallen

Das Rechnen mit reellen Zahlen ist durch unvermeidliche Rundungsfehler oft nicht genügend genau. Deswegen werden Berechnungen immer öfter mittels Intervallarithmetik implementiert. Dabei rechnet man statt mit der reellen Zahl a mit einem beschränkten Intervall $[a] = [a - \varepsilon, a + \varepsilon]$, wobei $\varepsilon > 0$ der Radius des Intervalls ist.
Alternativ kann man ein Intervall auch folgendermaßen definieren:

$$\mathbf{x} := [\underline{x}, \overline{x}],$$

wobei $\underline{x}$ bzw. $\overline{x}$ das Infimum bzw. das Supremum des Intervalls sind. Auf diese Weise berechnet man Schranken, die die exakte Lösung enthalten.

Die wichtigsten Funktionen für die Intervalle sind:

- $m(\mathbf{x})$ Mittelpunkt der Intervalls,
- $r(\mathbf{x})$ Radius des Intervalls,
- $\sup(\mathbf{x})$ Supremum des Intervalls,
- $\inf(\mathbf{x})$ Infimum des Intervalls und
- $is(\mathbf{x}, \mathbf{y})$ Durchschnitt zweier Intervalle.

Zwischen zwei Intervallen sind die elementaren Operationen $+$, $-$, $\times$ und $/$ folgen-

dermaßen definiert:

$$\begin{aligned}
\mathbf{x}+\mathbf{y} &= [\underline{x}+\underline{y}, \overline{x}+\overline{y}] \\
\mathbf{x}-\mathbf{y} &= [\underline{x}-\overline{y}, \overline{x}-\underline{y}] \\
\mathbf{x}\times\mathbf{y} &= [\min(\underline{xy}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{xy}), \max(\underline{xy}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{xy})] \\
1/\mathbf{x} &= [1/\overline{x}, 1/\underline{x}] \quad \text{für } 0 \notin \mathbf{x} \\
\mathbf{x}/\mathbf{y} &= \mathbf{x}\times 1/\mathbf{y}
\end{aligned}$$

Die Operation $\mathbf{x}/\mathbf{y}$ ist für $0 \in \mathbf{y}$ nicht definiert.
Um die Auswertung trotzdem durchführen zu können, erweitert man die Intervallarithmetik auch auf unendliche Intervalle.
Für zwei Intervalle $\mathbf{x}$ und $\mathbf{y}$ gilt:

$$\mathbf{x}/\mathbf{y} = \begin{cases}
\mathbf{x}\times 1/\mathbf{y} & \text{für } 0 \notin \mathbf{y} \\
[\overline{x}/\underline{y}, \infty] & \text{für } \overline{x} \leqslant 0 \text{ und } \overline{y} = 0 \\
[-\infty, \overline{x}/\overline{y}] \cup [\overline{x}/\underline{y}, \infty] & \text{für } \overline{x} \leqslant 0 \text{ und } \underline{y} < 0 < \overline{y} \\
[-\infty, \overline{x}/\overline{y}] & \text{für } \overline{x} \leqslant 0 \text{ und } \underline{y} = 0 \\
[-\infty, \infty] & \text{für } \underline{x} < 0 < \overline{x} \\
[-\infty, \underline{x}/\underline{y}] & \text{für } \underline{x} \geqslant 0 \text{ und } \overline{y} = 0 \\
[-\infty, \underline{x}/\underline{y}] \cup [\underline{x}/\overline{y}, \infty] & \text{für } \underline{x} \geqslant 0 \text{ und } \underline{y} < 0 < \overline{y} \\
[\underline{x}/\overline{y}, \infty] & \text{für } \underline{x} \geqslant 0 \text{ und } \underline{y} = 0 \\
\emptyset & \text{für } \underline{y} = \overline{y} = 0
\end{cases} .$$

Die Bildbereiche der elementaren Intervalloperatoren entsprechen dabei exakt den Bildbereichen der reellen Operatoren. Dies ändert sich jedoch, wenn man die Intervalloperationen zusammensetzt. Betrachtet man den Ausdruck $g(x) = x^2 - x$ und setzt für x das Intervall $[0{,}0.5]$ ein, so ist

$$[0{,}0.5]^2 - [0{,}0.5] = [0, 0.25] - [0, 0.5] = [-0.5, 0.25].$$

Der Bildbereich von g über $[0{,}0.5]$ ist aber $[-0.25{,}0]$. Somit wird im allgemeinem der Wert bei derZusammensetzung mehrerer verschiedener Intervalloperationen überschätzt.

Ein anderes Problem, das bei Intervallarithmetik entstehen kann, wird durch Rundungsfehler erzeugt. Betrachtet man das Beispiel

$$[0.1234, 0.4567] + [0.28739{,}0.45321] = \underbrace{[0.41079, 0.90991]}_{A},$$

so sieht man, dass bei gewöhnlicher Rundung auf vier signifikante Dezimalstellen das Ergebnis $[0.4108,0.9099] = B$ ist. Nun ist aber $A \not\subset B$, was die Inklusionsisotonie verletzt.
Um dies zu vermeiden, wird in der Intervallarithmetik das sogenannte Außenrunden verwendet. Das bedeutet, dass die untere Schranke des Intervalls nach unten und die obere Schranke nach oben gerundet wird. Somit ist das Ergebnis der obigen Rechnung $[0.4107, 0.9100] = C$, und es gilt $A \subset C$.

Weiters empfiehlt es sich beim Auswerten von linearen Konvexkombinationen, diese etwas umzuschreiben. Anstatt Rechnungen der Form

$$P = \alpha \cdot R_1 + (1 - \alpha)R_2$$

durchzuführen, ist es meistens günstiger

$$P = \alpha \cdot (R_1 - R_2) + R_2$$

zu berechnen, da es dadurch zu kleineren Überschätzungen durch das Rechnen mit den Intervallen kommt. Für genauere Details siehe [7].

Um mit Intervallarithmetik in Matlab zu rechnen, wurde von Prof. Dr. Siegfried M. Rump ein kostenloses Matlab-Tool namens Intlab [11] entwickelt, das eben dies ermöglicht.
Die Implementationen für diese Arbeit wurden mit Intlab Version 5.5 durchgeführt.

5.2 Intervall-Newton-Verfahren

Ein ganz wichtiges Verfahren, das im Zusammenhang mit Intervallarithmetik eine große Rolle spielt, ist das Intervall-Newton-Verfahren zur Suche von Nullstellen einer Funktion.
Das Verfahren findet, im Gegensatz zum einfachen Newton-Verfahren, fast immer alle Nullstellen der Funktion f im vorgegebenem Intervall.

Das einfache Newton-Verfahren ohne Intervall-Arithmetik versucht, um die Nullstellen der stetig differenzierbaren Funktion f zu bestimmen, diese Funktion lokal zu linearisieren. Die Tangente im Punkt $(\xi, f(\xi))$ hat die Gleichung

$$t(x) = f(\xi) + f'(\xi)(x - \xi)$$

und schneidet die X-Achse im Punkt

$$\hat{x} = \xi - \frac{f(\xi)}{f'(\xi)}.$$

Wählt man also einen Startpunkt x_0 aus dem Definitionsbereich von f, der „nahe genug“ an der Nullstelle der Funktion f liegt, so kommt man durch die Iteration

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \quad n > 1$$

schrittweise zu der Nullstelle $\bar{x}$.
Details über Vorteile, Nachteile und Konvergenzeigenschaften des Verfahrens kann man in jedem guten Buch über Numerik (zum Beispiel in [1]) nachschlagen.

Das Intervall-Newton-Verfahren betrachtet, anstatt einer Tangente an dem Punkt $P = (\xi, f(\xi))$ (für $\xi = \frac{a+b}{2} = m([a,b])$) des Definitionsbereichs $[a,b]$ von f, die Menge aller Geraden durch den Punkt P mit den Steigungen $f'([a,b]) := \{f'(x) | \ x \in [a,b]\}$.
Die Nullstellen aller dieser Geraden liegen in der Menge

$$N([a,b]) = \xi - \frac{f(\xi)}{f'([a,b])}. \tag{5.1}$$

Alle Nullstellen von f liegen im Definitionsbereich $[a,b]$. Bildet man also das Intervall $\mathcal{N}([a,b]) = [\min(N([a,b])), \max(N([a,b]))]$ und schneidet dieses mit dem Definitionsbereich, so engt man iterativ die Menge, die alle Nullstellen der Funktion f enthält, ein.
Setzt man also $[a,b]_0 = [a,b]$ so sieht die Iteration des Newton-Verfahrens wie folgt aus:

$$[a,b]_{n+1} = [a,b]_n \cap N([a,b]).$$

Abhängig von der Monotonie der Funktion f kann $N([a,b])$ aus einem Intervall oder zwei Intervallen (falls f nicht monoton) bestehen. Ist nämlich f nicht monoton, besitzt aber eine Nullstelle, so liegt die Null in der Menge $f'([a,b])$. Um $N([a,b])$ zu berechnen, muss man die erweiterte Intervallarithmetik bei der Intervall-Divison anwenden, wobei in manchen Fällen zwei Intervalle das Ergebnis sind.
Aus diesem Grund kann der Schnitt $[a,b]_{n+1} = [a,b]_n \cap N([a,b]_n)$

i. aus einem Intervall bestehen.

ii. aus zwei Intervallen bestehen.

iii. die leere Menge sein.

Tritt der Fall iii. ein, so hat die Funktion f keine Nullstelle im Bereich $[a,b]$.
Tritt der Fall ii. ein, so iteriert man mit jedem der beiden Intervalle einzeln.

Es besteht noch die Möglichkeit, dass $[a,b]_n \subset N([a,b]_n)$ und der Schnitt $[a,b]_{n+1} = [a,b]_n \cap N([a,b]_n)$ kein weiteres Einschränken des Intervalls darstellt. In diesem Fall ist keine Aussage über die Nullstelle möglich.

Alle diese Eigenschaften kann man in einem Satz zusammenfassen:

Satz 5.1 (Intervall-Newton-Verfahren)
Für eine stetige Intervall-Funktion $f : [a,b] \to \mathbb{R}$, sei f' deren Ableitung. Seien weiters $\mathbf{x} \subseteq [a,b]$ und $x \in \mathbf{x}$ und sei $N(\mathbf{x}) = x - \frac{f(x)}{f'(\mathbf{x})}$.

- Existiert eine Nullstelle x^* von f in $\mathbf{x}$, dann ist diese Nullstelle auch in $N(\mathbf{x})$ enthalten.
- Gilt $N(\mathbf{x}) \subset \mathbf{x}$, dann existiert eine Nullstelle von f in $\mathbf{x}$.
- Befindet sich in $\mathbf{x} = \mathbf{x}^0$ eine Nullstelle, dann ist diese Nullstelle ebenfalls in jedem $\mathbf{x}^k$ für $k = 1,2,3,\ldots$ enthalten, wobei $\mathbf{x}^k = \mathbf{x}^{k-1} \cap N(x^(k-1))$ ist.
- Falls der Schnitt $\mathbf{x} \cap N(\mathbf{x})$ die leere Menge bildet, dann existiert keine Nullstelle von f in $\mathbf{x}$.

Die Beweise dieser Aussagen findet man in [2, Kapitel 7] und teilweise auch in [6, Kapitel 8].

Das eigentliche Intervall-Newton-Verfahren, dessen genaue Implementation im letzten Kapitel angeführt ist, funktioniert nach folgendem Prinzip:

```
1  function Newton_Verfahren(f,f',tol,X,r)
2  %PseudoCode für das Intervall-Newton Verfahren
3  %f....Funktion dessen Nullstellen berechnet werden
4  %f'...Ableitung von f
5  %tol..eine Genauigkeitsgrenze
6  %X....Definitionsbereich an dem ausgewertet wird
7  %r....Anzahl der Unterteilungsbereiche für X
8  %Output
9  %E....Liste von Extremwerten

11 Unterteile X in r Teilintervalle
12 while (X~={})
13       U=Ein Teilintervall von X;
14       while rad(U)<tol
15         x=m[U];
```

```
16          berechne N(f,x)=x-f(x)/f'([a,b]);
17             berechne R=( Durchschnitt von N(f,x) und U);
18          if (R ist ein Intervall)
19                  U=R;
20             else (R sind zwei Intervalle R1 und R2)
21                  U=R1;
22                  R2 zu X dazufügen;
23             end
24       end
25       U zu E hinzufügen
26  end
```

Dabei sind folgende Argumente notwendig:

- `f` ist die Funktion deren Nullstellen gesucht werden. In der eigentlichen Implementation ist das die erste Ableitung der Komponentenfunktion.
- `f'` ist die Ableitung von `f`, somit die zweite Ableitung der Komponentenfunktion.
- `tol` bestimmt die Genauigkeit des Verfahrens.
- `X` ist der zu durchsuchende Definitionsbereich von `f`.
- `(r)` ist die Anzahl der Teilintervalle, in die `X` unterteilt wird, falls dieser Bereich ungeeignet ist (zu groß oder falls in diesem Bereich keine Aussage über Nullstellen möglich ist.)

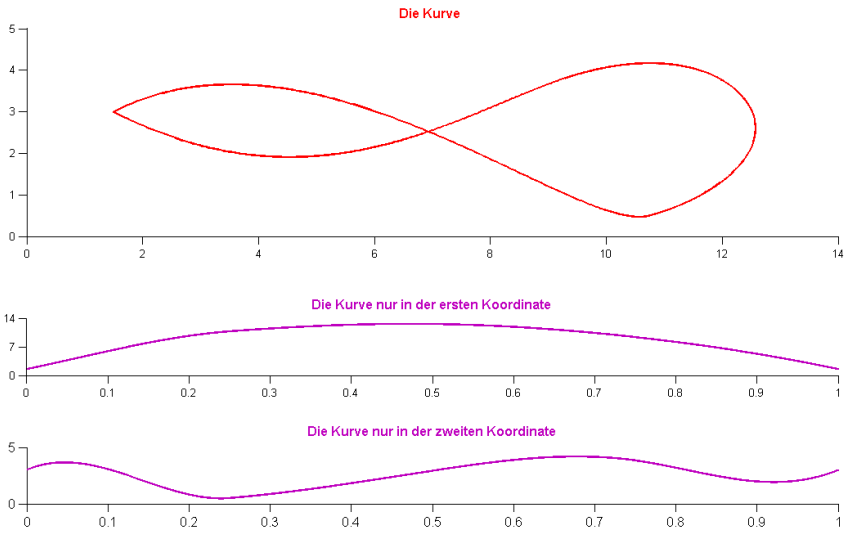
Zuerst wird das zu durchsuchende Intervall in `r` Teile zerlegt (falls notwendig), die dann einzeln dem Algorithmus übergeben werden, bis keine Teilintervalle mehr übrig sind. Leere Mengen werden dabei nicht beachtet. Wird eine Nullstelle entdeckt und ist der Radius des Intervalls kleiner als der Wert `tol`, so wird dieses Intervall zur Liste der Ergebnisse hinzugefügt. Ansonsten wird es weiter bis auf die gewünschte Genauigkeit verkleinert.

6 Intervall-Einschließungen von NURBS-Kurven

Das Ziel dieser Arbeit ist es Algorithmen zu implementieren, um eine minimale „Box“ um die Kurve legen zu können. Das bedeutet, dass man von allen vier Seiten den maximalen und den minimalen Wert der Kurve bestimmen muss. Die Vorgangsweise, diese Berechnung durchzuführen, wird in diesem Kapitel besprochen. Die zugehörigen Algorithmen sind im letzten Kapitel angeführt.

Um lineare Einschließungen von NURBS-Kurven zu implementieren, muss man zuerst ein Datenformat für die Kurven definieren.
Für alle unten angeführten Algorithmen müssen für jede NURBS-Kurve $C(u) = \frac{\sum_{i=0}^{n} N_{i,p}(u) P_i w_i}{\sum_{j=0}^{n} N_{j,p}(u) w_j}$ die $n+1$ Kontrollpunkte P_i in einem Spaltenvektor, der Knotenvektor U und die Gewichte w_i in einem Zeilenvektor vorhanden sein. Weiters muss man noch den Grad p der Kurve angeben, sowie n (die Anzahl der Punkte) und m (die Anzahl der Knoten).
Zusätzlich muss man noch den Wert `tol` festlegen. Dieser Wert bestimmt die Genauigkeit des Verfahrens.

Um den minimalen und den maximalen Wert jeder Seite berechnen zu können, muss man die Kurve $C(u)$ komponentenweise auswerten. Das bedeutet, die Kurve separat zu berechnen für jede Koordinate der Kontrollpunkte.



Auf die einzelnen Komponentenkurven $C_i(u)$ für $i = 1,2$ kann ein eindimensionales, numerisches Verfahren zur Berechnung von Minima und Maxima angewendet werden. Somit muss man für jede Dimension der Spline-Kurve den gesamten Algorithmus einmal durchführen.

In dieser Arbeit wurde mit dem Intervall-Newton-Verfahren gearbeitet.
Das Intervall-Newton-Verfahren ist ein numerisches Verfahren zur Berechnung der Nullstellen. Genauere Details dazu, kann man im Kapitel 5 nachlesen. Hier wird das Verfahren auf die Ableitungen der einzelnen Komponentenkurven $C_i'(u)$ für $i = 1,2$ angewendet. Zusätzlich braucht man noch die zweite Ableitung dieser Kurven, also die Werte $C_i''(u)$ für $i = 1,2$.

Der Aufruf der Implementation des Intervall-Newton-Verfahrens für die erste Komponentenkurve sieht folgendermaßen aus:

```
erg=inewtonKNRH(@Diff_intKNR1,@Diff_intKNR,na,p,U,P(:,1),w,X,tol,erg);
```

Die Argumente sind dabei

i. `Diff_intKNR1` ist der Algorithmus, der die erste Ableitung der Komponentenkurven auswertet.

ii. `Diff_intKNR` ist der Algorithmus, der die zweite Ableitung der Komponentenkurven auswertet.

iii. `n` ist die Anzahl der Kontrollpunkte.

iv. `p` ist der Grad der Kurve.

v. `U` ist der Knotenvektor.

vi. `P(:,1)` sind die ersten Koordinaten der Kontrollpunkte.

vii. `w` sind die Gewichte.

viii. `X` ist ein Teilintervall des Knotenvektors.

ix. `tol` ist der Genauigkeitswert.

x. `erg` ist eine Liste mit schon gefundenen Extremwerten.

Die Argumente ii. - vi. definieren die NURBS-Kurve.
Das letzte Argument ist optional und kann auch ausgelassen werden. (Der Standardwert ist in diesem Fall die leere Matrix).

Zwischen den einzelnen Knoten ist die NURBS-Kurve unendlich oft differenzierbar. Existieren aber Knoten, deren Ordnung größer als `p` ist, so muss die Kurve an diesen Stellen nicht differenzierbar sein. Diese Fälle haben zwei Konsequenzen:

- Man sollte dem Intervall-Newton-Verfahren höchstens ein Teilintervall des Knotenvektors übergeben, außer man weiß vorher, wo die nicht differenzierbaren Stellen liegen.
- Da die nicht differenzierbaren Knoten vom Intervall-Newton-Verfahren übersprungen werden, muss man diese zur Liste der Nullstellen hinzufügen.

Die Auswertung der linearen Einschließung besteht aus folgenden Schritten

i. zerlegen der NURBS-Kurve in ihre Komponentenkurven,

ii. eine Liste `L` bilden, die die Endpunkte von `U` und die nicht differenzierbaren Stellen (falls welche existieren) enthält,

iii. eine Liste `Z` bilden, die die nichtleeren Teilintervalle des Knotenvektors `U` beinhaltet,

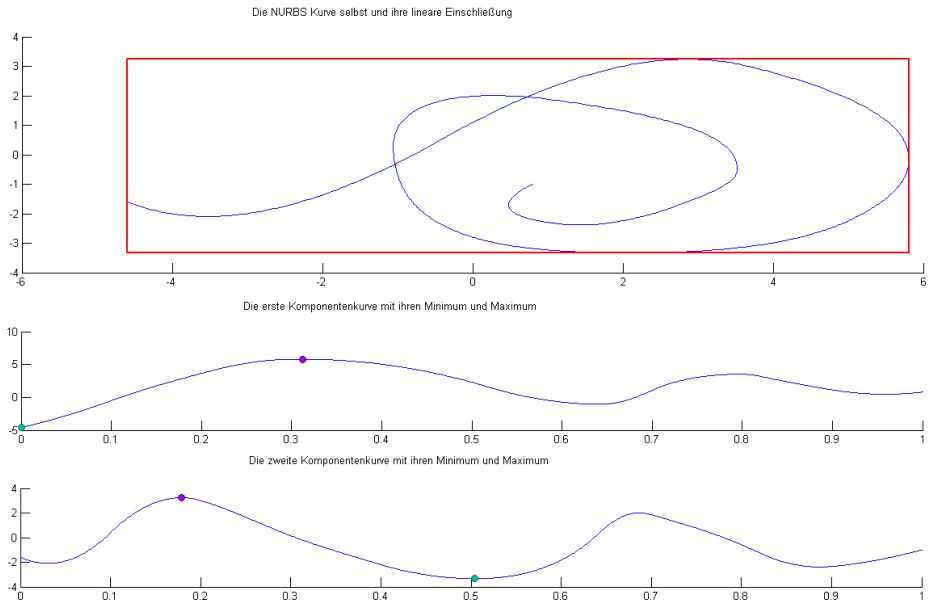
iv. für eine der Komponentenkurven und auf jedes einzelne Teilintervall aus `X` das Intervall-Newton-Verfahren anwenden und gefundene Nullstellen zu der Liste `L` hinzufügen,

v. die Werte aus der Liste `L` auswerten und das Minumum und das Maximum dieser Auswertung bestimmen und speichern,

vi. die Schritte vier und fünf auf die nächste Komponentenkurve anwenden,

vii. usw. ...

Der Aufruf des Algorithmus 'main', dessen Implementierung ebenfalls im Anhang steht, fasst alle diese Vorgänge zusammen.
Zuerst wird mit dem Algorithmus 'suchbereich' ein Suchbereich definiert. Dabei werden die zwei äußersten Kontrollpunkte von oben und von unten und deren Indexzahlen bestimmt. Seien die Indizes i, j, k und l mit $i < j < k < l$. Der Suchbereich ist dann $[U(i), \ldots, U(i+p+1), U(j), \ldots, U(j+p+1), \ldots, U(l+p+1)]$; Dann wird die Liste mit den nicht differenzierbaren Stellen der Kurve (falls vorhanden) gebildet und daraus wird das Minimum und das Maximum heraus gesucht.
Als nächstes wendet man das Intervall-Newton-Verfahren auf die erste Kmponentenkurve und erstellt die Liste der Extremwerte. Anschließend werden dieser Liste die Endpunkte und das Minimum und Maximum der Liste der nicht differenzierbaren Stellen der Kurve hinzugefügt. Schließlich wird aus dieser Liste das globale Minimum und das globale Maximum herausgenommen.
Dieselbe Vorgangsweise wird der zweiten Komponentenkurve zuteil, wobei man das Minimum und das Maximum an den nicht differenzierbaren Stellen nicht wieder berechnen musss.
Im Anschluss gibt es noch eine grafische Darstellung der Kurve selbst, mit der Box und den Extremwerten der Komponentenkurven, wie es im folgenden Bild dargestellt wird.



Das ganze Verfahren zur linearen Einschließung hängt also von der Dimension der Kurve und von der Anzahl der unterschiedlichen Knoten im Knotenvektor U ab. Je höher beide Werte sind, desto ungeeigneter ist der Algorithmus.

Das Herzstück des Algorithmus, das Intervall-Newton-Verfahren, gliedert sich in zwei Teile, die Berechnung der Ableitungen und die Newton-Iteration selbst.
Die Berechnung der Ableitungen ist verhältnismäßig aufwendig, da der Algorithmus 'Diff_Int' bei jeder Iteration des Intervall-Newton-Verfahren zweimal ausgeführt wird. Einmal um $C'(m([u_i, u_{i+1}]))$ zu berechnen und das zweite Mal um $C'([u_i, u_{i+1}])$ zu berechnen.
Die Newton-Iteration braucht meist nur wenige Schritte um eine Nullstelle zu lokalisieren.

Somit steigt auch mit größerer Anzahl der Kontrollpunkte auch der Aufwand, zur Berechnung der linearen Einschließung stark, weil der Aufwand für das Differenzieren steigt.

Folgende Berechnungen wurden für eine eindimensionale Kurve vom Grad $p = 3$ und steigender Knotenanzahl durchgeführt.

Anzahl der Knotenpunkte	Gesamtzeit in Sekunden	Zeit für die Ableitungen	Differenz	Anzahl der Newton-Iterationen im Durchschnitt
$n = 4$	0.1864	0.1466	0.0402	1
$n = 5$	0.4081	0.3927	0.0517	1
$n = 6$	5.0832	4.9275	0.1557	4
$n = 10$	6.2673	5.9185	0.3488	3
$n = 20$	18.2956	17.8586	0.4261	2
$n = 50$	26,2364	27,57925	0.6435	2
$n = 70$	168,2395	166,4215	1.8180	1
$n = 100$	303,7653	301,0075	2.7578	2

Man muss erwähnen, dass mit steigender Größe der Spline, eine einzelne Iteration des Newtonverfahrens immer weniger Zeit braucht, da die zu durchsuchenden Bereiche immer kleiner werden.

Die Abhängigkeit des Verfahrens vom Grad der NURBS-Kurve wirft ein besseres Licht auf das Verfahren zur Einschließung. Die folgende Tabelle zeigt die Werte für eine NURBS-Kurve mit 10 Punkten:

Der Grad der Kurve	Gesamtzeit in Sekunden	Zeit für die Ableitungen	Differenz	Anzahl der Newton-Iterationen im Durchschnitt
$p = 2$	15,5633	15,2486	0.3153	3
$p = 3$	18,5922	18,174	0.4182	6
$p = 4$	2,2887	2,2294	0.0592	1
$p = 5$	2,2617	2,2212	0.0405	1
$p = 6$	2,2017	2,161	0.0407	1
$p = 7$	2,0157	2,0396	0.0192	1
$p = 8$	1,8101	1,7197	0.0182	1
$p = 9$	1,4035	1,3917	0.0118	1
$p = 10$	0,8234	0,8157	0.0077	1

Der Aufwand verkleinert sich aus zwei Gründen:

i. Durch die steigende Größe des Grades verringert sich die Anzahl der unterschiedlichen Knoten im Knotenvektor, womit sich die Anzahl den Newton - Aufrufe ebenfalls verringert.

ii. Da der Grad steigt, wird die Kurve „glatter“ und benötigt weniger Newtoniterationen, um die Nullstelle zu berechnen.

Alle Zahlen wurden mit einem 16-Bit Windows PC berechnet mit einem Intel Pentium Dual CPU T2390 @1.58 GHz und 2GB RAM.

7 Implementationen

Dieses letzte Kapitel enthält alle Implementationen, die zum Auswerten der Box-Einschließungen von NURBS-Kurven, wie es Kapitel 6 beschrieben wird, benötigt werden. Zusätzlich, im zweiten Unterabschnitt, werden noch kleine Hilfsalgorithmen angegeben, die hauptsächlich für das Rechnen mit Intervallen benötigt werden, aber auch einige, die für die NURBS Algorithmen benötigt werden.
Zum Schluss sind noch wenige nützliche Implementationen angeführt; unter anderem auch Algorithmen zur graphischen Darstellung der Kurven.

7.1 Algorithmen zur Box-Einschließung

7.1.1 Auswertung und Ableitung

Die Auswertung der NURBS-Kurven wird hauptsächlich mithilfe des Algorithmus 'CurvePntByCornerCut' durchgeführt. Der Algorithmus gibt die Werte $C(u)$ und $C^w(u)$ aus, also den Wert der NURBS-Kurve und denselben Wert in homogenen Koordinaten.

```
1  function [C Cw]=CurvePntByCornerCut(n,p,U,P,w,u)
2  %[C Cw]=CurvePntByCornerCut(n,p,U,P,w,u)
3  %Berechnet Punkt auf der Kurve durch den
4  %de Boor Algorithmus
5  %n....Anzahl der Punkte (0-n)
6  %p....Grad der Spline
7  %U....Knotenvektor
8  %P....Kontrollpunkte
9  %w....Gewichte (falls keine Gewichte
10 %     w=[] setzen)
11 %u....Punkt an dem Ausgewertet wird
12 %Outputs:
13 %C....Punkt an der Stelle u
14 %Cw...Punkt an der Stelle u in
15 %     homogenen Koordinaten
```

```
18 if (size(w)==[0 0])
19     w=ones(1,n+1);
20 end
21 Pw=H(P,w); C=[];
22 sizePw=size(Pw);

24 %SpeziallFall: Endpunkte
25     %Für B-Splines
26 if sizePw(2)==1
27     if(u==U(1))
28         C=Pw(1); Cw=[C w(1)];
29         return;
30     end
31     if(u==U(n+p+2))
32         C=Pw(n+1); Cw=[C w(n+1)];
33         return;
34     end
35 else
36     %und für NURBS
37     if(u==U(1))
38         for i=1:sizePw(2)-1
39             a=Pw(1,:);  C=[C a(i)/a(end)];
40             Cw=a;
41         end
42         return;
43     end
44     if(u==U(n+p+2))
45         for i=1:sizePw(2)-1
46             a=Pw(n+1,:); C=[C a(i)/a(end)];
47             Cw=a;
48         end
49         return;
50     end
51 end

53 [k, s]=FindSpanMult(n,p,u,U);
54 r=p-s;
55 if r<0
56     [C Cw]=CurvePoint2(n,p,U,P,w,u);
57     return;
58 end
59 for i=0:r
60     Rw(i+1,:)=Pw(k-p+i+1,:);
```

```
61 end

63 for j=1:r
64     for i=0:r-j
65         alfa=(u-U(k-p+j+i+1))/(U(i+k+2)-U(k-p+j+i+1));
66         Rw(i+1,:)=alfa*(Rw(i+2,:)-Rw(i+1,:))+Rw(i+1,:);
67     end
68 end;

70 sizeRw=size(Rw);
71 Cw=Rw(1,:);
72 if (sizeRw(2)==1)
73     C=Rw(1);
74 else
75     C=Hinv(Rw(1,:));
76 end
```

Der Algorithmus funktioniert auch für B-Splines indem man für Gewichte den Wert [ ] setzt. In diesem Fall wird ein Zeilenvektor von Gewichten gebildet, wobei jedes Gewicht den Wert 1 erhält.

Die Implementation von der Intervall-Erweiterung dieses Algorithmus ist der Algorithmus 'CurvePntByCornerCut_int'. Dieser sieht folgendermaßen aus:

```
1  function [C Cw]=CurvePntByCornerCut_int(n,p,U,P,w,u)
2  %[C Cw]=CurvePntByCornerCut(n,p,U,P,w,u)
3  %Berechnet Punkt auf der Kurve durch den
4  %de Boor Algorithmus
5  %INTVAL-Version
6  %n....Anzahl der Punkte (0-n)
7  %p....Grad der Spline
8  %U....Knotenvektor
9  %P....Kontrollpunkte
10 %w....Gewichte (falls keine Gewichte w=[] setzen)
11 %u....Punkt an dem Ausgewertet wird
12 %Outputs
13 %C....Punkt an der Stelle u
14 %Cw...Punkt an der Stelle u in homogenen Koordinaten

16 sizeP=size(P);
17 %Falls eine B-Spline Kurve
18 if (size(w)==[0 0])
19     w=[];
20     w=intval(w);
```

```
21      for i=1:sizeP(1)
22          w=[w intval(1)];
23      end
24  end

26  Pw=H(P,w); C=[]; C=intval(C);
27  sizePw=size(Pw);

29  if sizePw(2)==1
30      if(u==U(1))
31          C=Pw(1);
32          Cw=[C w(1)];
33          return;
34      end
35      if(u==U(n+p+2))
36          C=Pw(n+1);
37          Cw=[C w(n+1)];
38          return;
39      end
40  else
41      if(u==U(1))
42          for i=1:sizePw(2)-1
43              a=mid(Pw(1,:));
44              hv=a(i)/a(end);
45              C=[C hv]; Cw=Pw(1,:);
46          end
47          return;
48      end
49      if(u==U(n+p+2))
50          for i=1:sizePw(2)-1
51              a=mid(Pw(n+1,:));
52              hv=a(i)/a(end);
53              C=[C hv]; Cw=Pw(n+1,:);
54          end
55          return;
56      end
57  end

59  [k, s]=FindSpanMult(n,p,u,U);
60  r=p-s;
61  if r<0
62      [C Cw]=CurvePoint2(n,p,U,P,w,u);
63      return;
64  end
```

```
65 for i=0:r
66     Rw(i+1,:)=Pw(k-p+i+1,:);
67 end
68 for j=1:r
69      for i=0:r-j
70         if ((U(i+k+1+1)-U(k-p+j+i+1))==0)
71             alfa=intval(0);
72         else
73            alfa=(u-U(k-p+j+i+1))/(U(i+k+1+1)-U(k-p+j+i+1));
74         end
75         Rw(i+1,:)=alfa*(Rw(i+1+1,:)-Rw(i+1,:))+Rw(i+1,:);
76     end
77 end;
78 sizeRw=size(Rw); Cw=Rw(1,:);

80 if (sizeRw(2)==1)
81     C=Rw(1);
82 else
83     for kk=1:sizeRw(2)-1
84         hv=divideKNR(Rw(1,kk),Rw(1,end));
85         C=[C divideKNR(Rw(1,kk),Rw(1,end))];
86     end
87 end
```

Dieser Algorithmus berechnet analog zu 'CurvePntByCornerCut', die Ergebnisse sind jedoch Intervalle.

Will man die Kurve an einem Knoten auswerten, dessen Ordnung $\geqslant p+1$ ist, so kann man den 'CurvePntByCornerCut'-Algorithmus nicht verwenden. Tritt so ein Fall ein, so werden diese Stellen mit dem Algorithmus 'CurvePoint2' berechnet.

```
1  function [C Cw]=CurvePoint2(n,p,U,P,w,u)
2  %[C Cw]=CurvePoint2(n,p,U,P,w,u)
3  %Berechnet den Wert einer rationalen B-Spline bei u
4  %n....Anzahl der Punkte (0-n)
5  %p....Grad der Spline
6  %U....Knotenvektor
7  %P....Kontrollpunkte
8  %w....Gewichte (falls keine Gewichte
9  %     w=[] setzen)
10 %u....Punkt an dem Ausgewertet wird
11 %Outputs
12 %C....Der Punkt an der Kurve
```

```
13  %Cw...Der Punkt an der Kurve in homogenen Koordinaten

15  if (size(w)==[0 0])
16      w=ones(1,n+1);
17  end

19  Pw=H(P,w);
20  i=FindSpanMult(n,p,u,U);
21  %Berechnet die Basisfunktionen
22  N(1)=1;
23  for j=1:p
24      saved=0;
25      left(j)=u-U(i-j+2);
26      right(j)=U(i+j+1)-u;
27      for r=0:j-1
28          temp=N(r+1)/(right(r+1)+left(j-r));
29          if (isnan(temp)|| isinf(temp))
30              temp=0;
31          end
32          a=saved+right(r+1)*temp;
33          N(r+1)=a;
34          saved=left(j-r)*temp;
35      end
36      N(j+1)=saved;
37  end
38  %Berechnet den Punkt.
39  Cw=0;
40  for j=0:p
41       Cw=Cw+N(j+1)*Pw(i-p+j+1,:);
42  end
43  C=Hinv(Cw(1,:));
```

Wie im Unterabschnitt 4.1 beschrieben, werden hierbei zuerst alle $p+1$ für die Stelle u relevanten Basisfunktionen berechnet; *(Zeile 17-32)* dann werden sie mit den zugehörigen Kontrollpunkten multipliziert, und anschließend wird die Summe gebildet *(Zeile 35-37)*.
Die Ausgabe besteht auch hier aus dem Wert der NURBS-Kurve an der Stelle u und demselben Wert in homogenen Koordinaten. Dieser Algorithmus kann ebenfalls für B-Spline-Kurven angewendet werden, indem man für die Gewichte eine leere Matrix [ ] einsetzt.
'CurvePoint2' benötigt keine eigenständige Intervallversion; falls die übergebenen Argumente Intervalle sind, wird auch als Ergebnis ein Intervall ausgegeben.

Der Algorithmus 'Diff' berechnet die erste und die zweite Ableitung nach den Formel (3.10), wobei $A'(u)$ und $A''(u)$ mit den Formeln (2.4) und (2.5) ausgewertet werden.

```
1  function [C CC]=Diff(p,n,m,P,w,U,u)
2  %C=Diff(p,n,m,P,w,U,u)
3  %Berechnet die Ableitung der NURBSKurve  am Punkt u
4  %n....Anzahl der Punkte (0-n)
5  %p....Grad der Spline
6  %U....Knotenvektor
7  %P....Kontrollpunkte
8  %m....Anzahl der Elemente von U [0...m]
9  %w....Gewichte
10 %u....Punkt an dem Ausgewertet wird
11 %Outputs
12 %C....Erste Ableitung
13 %CC...Zweite Ableitung


16 np=n-1;
17 [A Cw]=CurvePntByCornerCut(n,p,U,P,w,u);
18 wu=Cw(end); B=A;
19 A=Cw(1:end-1);

21 nU=U(2:end-1);
22 Pw =H(P,w); Qw=[];
23 for i=0:n-1
24     hv=(nU(i+p+1)-nU(i+1));
25     if hv==0
26         Qw(i+1,:)=0;
27     else
28         Qw(i+1,:)=p*(Pw(i+2,:)-Pw(i+1,:))/hv;
29     end
30 end
31 [Q,wq]=Hinv(Qw);
32 [NO Cpw]=CurvePntByCornerCut(np,p-1,nU,Q,wq,u);
33 Dwu=Cpw(end); DA=Cpw(1:end-1);

35 %Erste Ableitung:
36 C=DA/wu-Dwu*A/(wu^2);

38 nnU=nU(2:end-1);
39 QQw=[]; q=p-1;
40 for i=0:n-2
41     hv=(nnU(i+q+1)-nnU(i+1));
```

```
42      if hv==0
43          QQw(i+1,:)=0;
44      else
45          QQw(i+1,:)=q*(Qw(i+2,:)-Qw(i+1,:))/hv;
46      end
47  end
48  [QQ wqq]=Hinv(QQw); nnp=np-1;
49  [NO Cqw]=CurvePntByCornerCut(nnp, q-1, nnU, QQ, wqq, u);
50  DDwu=Cqw(end); DDA=Cqw(1:end-1);

52  %Zweite Ableitung:
53  CC=DDA/wu-DA*Dwu/(wu^2)-C*Dwu/wu-B*DDwu/wu+B*(Dwu^2)/(wu^2);
```

Zuerst werden aus den Knotenvektoren die Endpunkte entfernt (für die erste Ableitung Zeile 21 und für die zweite Ableitung Zeile 39). Die Funktionen $g(u)$ und $A(u)$ erhält man durch das Ausführen von 'CurvePntByCornerCut' (*Zeile 17*). $g(u)$ ist die letzte Koordinate der Ausgabe in homogenen Koordinaten und $A(u)$ enthält die restlichen Koordinaten. Ebenso erhält man $A'(u)$, $A''(u)$ sowie $g'(u)$, $g''(u)$ durch das Ausführen desselben Algorithmus (*Zeile 32 bzw. 49*), wobei man aber den Grad sowie die Anzahl der neuen Knoten und der Kontrollpunkte entsprechend verkleinern muss.

Der Ableitungsalgorithmus für Intervalle, 'Diff_intKNR', arbeitet nach dem selben Prinzip wie 'Diff'.

```
1   function [C CC]=Diff_intKNR(p,n,m,P,w,U,u)
2   %C=Diff(p,n,m,P,w,U,u)
3   %Berechnet die Ableitung der NURBSKurve  am Punkt u
4   %INTVAL-Version erwiterte Intervalle
5   %n....Anzahl der Punkte (0-n)
6   %p....Grad der Spline
7   %U....Knotenvektor
8   %P....Kontrollpunkte
9   %m....Anzahl der Elemente von U [0...m]
10  %w....Gewichte
11  %u....Punkt an dem Ausgewertet wird
12  %Outputs
13  %C....Erste Ableitung
14  %CC...Zweite Ableitung

16  np=n-1;
17  [A Cw]=CurvePntByCornerCut_int(n,p,U,P,w,u);
```

```
wu=Cw(end);
B=A; A=Cw(1:end-1);

nU=U(2:end-1);
Pw=H(P,w); Qw=[];
Qw=intval(Qw);
for i=0:n-1
    hv=(nU(i+p+1)-nU(i+1));
    if hv==infsup(0,0)
        QQ(i+1,:)=infsup(0,0);
    else
        Qw(i+1,:)=p*(Pw(i+2,:)-Pw(i+1,:))/hv;
    end
end
[Q,wq]=Hinv(Qw);
[Cp Cpw]=CurvePntByCornerCut_int(np,p-1,nU,Q,wq,u);
Dwu=Cpw(end);
DA=Cpw(1:end-1);

%Erste Ableitung
wu2=factorKNR(wu,2);
hv1=divideKNR(DA,wu);
hv2=divideKNR(Dwu*A,wu2);
C=hv1-hv2;


nnU=nU(2:end-1);
QQw=[]; QQw=intval(QQw);
q=p-1;
for i=0:n-2
    hv=(nnU(i+q+1)-nnU(i+1));
    if hv==infsup(0,0)
        QQw(i+1,:)=infsup(0,0);
    else
        QQw(i+1,:)=q*(Qw(i+2,:)-Qw(i+1,:))/hv;
    end
end
[QQ wqq]=Hinv(QQw); nnp=np-1;
[Cq Cqw]=CurvePntByCornerCut_int(nnp, q-1, nnU, QQ, wqq, u);
DDwu=Cqw(end); DDA=Cqw(1:end-1);

%Zweite Ableitung
Dwu2=factorKNR(Dwu,2);
hv1=divideKNR(DDA,wu);
```

```
62 hv2=divideKNR(DA*Dwu,wu2);
63 hv3=divideKNR(C*Dwu,wu);
64 hv4=divideKNR(B*DDwu,wu);
65 hv5=divideKNR(B*Dwu2,wu2);
66 CC=hv1-hv2-hv3-hv4+hv5;
```

7.1.2 Intervall-Newton-Verfahren

Diese Implementation des Intervall-Newton-Verfahrens für NURBS-Kurven arbeitet nach dem Prinzip aus Kapitel 5.2. Die Implementation ist aber doch etwas komplexer als dort dargestellt wird.

```
1  function erg=inewtonKNRH(f,ff,n,p,U_int,P_int,w_int,X,tol,erg)
2  %Interval Newton-Verfahren
3  %ff.....Ableitungsfunktion
4  %n......Anzahl der Punkte (0-n)
5  %p......Grad der Spline
6  %U......Knotenvektor
7  %P......Kontrollpunkte
8  %w......Gewichte
9  %X......zuuntersuchender Definitionsbereich
10 %tol....Genauigkeitswert
11 %erg....(Optional) Liste mit Extremwerten
12 %Output
13 %erg....Liste mit Extremwerten

15 if nargin==8
16     erg=[];
17 end

19 m=n+p+1;
20 L=X;
21 zugef=0;
22 while ~(isempty(L))
23     Xm=L(end);
24     L=L(1:(end-1));
25     count=0;
26     while (rad(Xm)>tol && count<20)
27         count=count+1;
28         x=mid(Xm);
29         fx=f(p,n,m,P_int,w_int,U_int,x);
30         [NO F]=ff(p,n,m,P_int,w_int,U_int,Xm);
```

```
31          N=x-divideKNR(fx,F);
32          Xn=intersect(X,N);
33          shp=size(Xn); shp=shp(1);  Xnn=[];  %NaNs raus
34          for i=1:shp
35              if (~isnan(Xn(i)))
36                  Xnn=[Xnn; Xn(i)];
37              end
38          end
39          Xn=Xnn; shp2=size(Xn); shp2=shp2(1);
40          if (shp2==0)
41              %display('leer');
42              if(zugef==1)
43                  Xm=infsup(NaN,NaN);
44              else
45                  h=rad(X)/p;
46                  L=[];
47                  start=sup(X);
48                  for i=1:(2*p)
49                      hv=infsup(start-h,start);
50                      L=[L;hv];  start=start-h;
51                  end
52                  Xm=infsup(inf(X),sup(L(end)));
53                  L=L(1:end-1);
54                  zugef=1;
55              end
56          end
57          if (shp2==1)
58              %display('einschränkung')
59              hv=intersect(Xm, Xn);
60              if isnan(hv)
61                 if (inf(Xn)==inf(X))
62                     Xm=infsup(NaN,NaN);
63                 else
64                     Xm=Xn;
65                 end;
66              else
67                 Xm=hv;
68              end
69          end
70          if (shp2==2)
71              %display('aufteilen')
72              teile=intersect(Xn(1),Xn(2));
73              if isnan(teile)
74                  if (zugef==0)
```

```
75                      %display('zugef ist 0')
76                      L=Xn(1);
77                      zugef=1;
78                      Xm=infsup(inf(Xn(2)),mid(Xn(2)));
79                      hv=infsup(mid(Xn(2)),sup(Xn(2)));
80                      L=[L;hv];
81                  else
82                      %display('zugef ist 1')
83                      lL=length(L);
84                      if (lL==0)
85                           L=Xn(1);
86                      else
87                           coll=[];
88                           for i=1:lL
89                               hv=intersect(L(i),Xn(1));
90                               if ~isnan(hv)
91                                   coll=[coll; hv];
92                               end
93                           end
94                          L=coll;
95                      end
96                      Xm=Intersect(Xm,Xn(2));
97                  end
98              else
99                Xm=teile;
100             end
101         end
102     end
103     if (~isnan(Xm) && rad(Xm)<tol)
104         counts=[counts; count];
105         BB=ismember(Xm,erg);
106         if all(BB==0)
107             erg=[erg; Xm];
108         end
109     end
110 end
```

Leere Mengen werden ausgeschlossen.
Die Funktion `f` ist der Algorithmus 'Diff_intKNR' bis zur Zeile 41, da an dieser Stelle die zweite Ableitung nicht benötigt wird.

Schließlich wird noch der Algorithmus zum Ausführen aller Implementationen für eine vorgegebene NURBS-Kurve und einen Genauigkeitswert `tol` angeführt.

```
function main(n,p,P,w,U,tol)
%führt alles aus


m=n+p+1;
P_int=intval(P);
w_int=intval(w);
U_int=intval(U);


%Undifferenzierbarkeitsbereiche + Endpunkte
extremws=[];
for i=p+1:m-p
    [k, s]=FindSpanMult(n,p,U(i),U);
    if (s>=p)
        extremws=[extremws; intval(U(i))];
    end
end
extremws=[extremws; intval(U(1)); intval(U(end))];



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%X-Koordinate%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
display('%%%%%%%%%%%%%%%%%% X-Koordinate %%%%%%%%%%%%%%%%')
NU=suchbereich(P(:,1),U,m,p);
extremw=[];
lNU=length(NU);
all=tic;
einzeln=0;
for i=1:lNU-1
    X=infsup(NU(i),NU(i+1));
    erg=[];
    erg=inewtonKNRH(@Diff_intKNR1,@Diff_intKNR,n,p,U_int,P_int(:,1),w_int,X,tol,erg);
    if ~isempty(erg)
        rad(erg);
        extremw=[extremw; erg];
    end
end
toc(all)
%Endpunkte und Undifferenzierbarkeitsstellen dazufügen

extremw=[extremw; extremws];
```

```
45 ss=length(extremw);
46 Ext=[];
47 for i=1:ss
48     C=CurvePntByCornerCut_int(n,p,U_int,P_int(:,1),w_int,extremw(i,1));
49     Ext=[Ext; C];
50 end

52 %Minimum und Supremum suchen und speichern
53 Exts=sup(Ext(:,1));
54 Exti=inf(Ext(:,1));
55 [supx index]=max(Exts);
56 supx=[mid(extremw(index)),supx];
57 [infx index]=min(Exti);
58 infx=[mid(extremw(index)),infx];



62 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
63 %%%%%%%%%%%%%%%%%%%%%%%%%Y-Koordinate%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
64 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
65 display('%%%%%%%%%%%%%%%%%% Y-Koordinate %%%%%%%%%%%%%%%')
66 NU=suchbereich(P(:,2),U,m,p);
67 extremw=[];
68 lNU=length(NU);
69 all=tic;
70 for i=1:lNU-1
71     X=infsup(NU(i),NU(i+1));
72     erg=[];
73     erg=inewtonKNRH(@Diff_intKNR1,@Diff_intKNR,n,p,U_int,P_int(:,2),w_int,X,tol,erg);
74     if ~isempty(erg)
75         rad(erg);
76         extremw=[extremw; erg];
77     end
78 end
79 toc(all)



82 %Endpunkte und Undifferenzierbarkeitsstellen dazufügen
83 extremw=[extremw; extremws];
84 ss=length(extremw);
85 Ext=[];
86 for i=1:ss
87     C=CurvePntByCornerCut_int(n,p,U_int,P_int(:,2),w_int,extremw(i,1));
88     Ext=[Ext; C];
```

```
89  end

91  %Minimum und Supremum suchen und speichern
92  Exts=sup(Ext(:,1));
93  Exti=inf(Ext(:,1));
94  [supy index]=max(Exts);
95  supy=[mid(extremw(index)),supy];
96  [infy index]=min(Exti);
97  infy=[mid(extremw(index)),infy];


100 display('%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% PLOT %%%%%%%%%%%%%%%%')
101 subplot(4,1,1:2)
102 hold on;
103 zeichneNURBSKurve(n,p,U,P,w,'b');
104 line([infx(2), infx(2)],[infy(2), supy(2)],'color','r');%links
105 line([supx(2), supx(2)],[infy(2), supy(2)],'color','r');%rechts
106 line([infx(2), supx(2)],[supy(2), supy(2)],'color','r');%oben
107 line([infx(2), supx(2)],[infy(2), infy(2)],'color','r');%unten
108 hold off;
109 subplot(4,1,3)
110 hold on
111 zeichneNURBSKurve(n,p,U,P(:,1),w,'b');
112 scatter(supx(1),supx(2))
113 scatter(infx(1),infx(2))
114 hold off
115 subplot(4,1,4)
116 hold on
117 zeichneNURBSKurve(n,p,U,P(:,2),w,'b');
118 scatter(supy(1),supy(2))
119 scatter(infy(1),infy(2))
120 hold off
```

7.1.3 Kleinere Hilfsalgorithmen

Die hier aufgelisteten Algorithmen sind zusätzliche Implementationen, die für die oberen Algorithmen immer wieder benötigt werden.

Die perspektive Abbildung H aus der Gleichung (3.7) wandelt die Kontrollpunkte und deren Gewichte in homogene Koordinaten um.

```
1  function Pw=H(P,w)
2  %Pw=H(P,w)
```

```
3  %Fügt Punkte in einem Array und Gewichte
4  %zu homogenen Koordinaten zusammen

6  sizeP=size(P);
7  if (sizeP==[0 0])
8      Pw=w'; return
9  end

11 Pw=[]; n=sizeP(1);
12 dim=sizeP(2);
13 for i=1:n
14     helpvar=[];
15     if (w(i)~=0)
16     for j=1:dim
17         helpvar=[helpvar P(i,j)*w(i)];
18     end
19     helpvar=[helpvar w(i)];
20     Pw=[Pw; helpvar];
21     else
22     Pw=[Pw; P(i,:) w(i)];
23     end;
24 end
```

Die Inverse der perspektiven Abbildung H trennt die in homogenen Koordinaten gegebenen Kontrollpunkte in Punkte und deren Gewichte.

```
1  function [P,w]=Hinv(Pw)
2  %[P,w]=Hinv(Pw)
3  %Trennt 2 und 3 koordinatige Punkte von Homogenen Koordinaten
4  %Punkte in einem Array

6  sizePw=size(Pw);
7  if (sizePw(2)==1)
8      P=[]; w=Pw'; return
9  end

11 dim=sizePw(2)-1;
12 w=Pw(:,end); w=w';
13 Phelp=Pw(:,1:dim); P=[];

15 for i=1:sizePw(1)
16     helpvar=[];
17     if (Pw(i,end)~=0)
```

```
18      for j=1:dim
19          helpvar=[helpvar Phelp(i,j)/w(i)];
20      end
21      else
22      helpvar=Pw(i,1:dim);
23      end
24      P=[P; helpvar];
25  end
```

Der folgende Algorithmus gibt an, in welchem Teilintervall des Knotenvektors sich ein Punkt u des Definitionsbereichs befindet. Außerdem gibt er die Ordnung des Punktes an, falls er ein Knoten ist. Die Implementation ist eine Erweiterung eines Algorithmus aus [8].

```
1   function [k, s]=FindSpanMult(n,p,u,U)
2   %Berechnet den Knotspanindex k
3   %und die Multiplizität s

5   sizeU=size(U);
6   if (u>U(end))
7       display('Punkt oberhalb vom Knotvektor')
8       k='falsch'; s='falsch'; return
9   end
10  if (u<U(1))
11      display('Punkt unterhalb vom Knotvektor')
12      k='falsch'; s='falsch'; return
13  end

15  if (u==U(n+2))
16      k=n; s=0;
17      for t=1:sizeU(2)
18          if u==U(t)
19              s=s+1;
20          end
21      end
22      return;
23  end;

25  low=p; high=n+1;
26  mid=fix((low+high)/2);

28  while (u<U(mid+1) | u>=U(mid+2))
29      if (u<U(mid+1))
```

```
30          high=mid;
31      else
32          low=mid;
33      end;
34      mid=fix((low+high)/2) ;
35  end;
36  k=mid;

38  s=0;
39  for t=1:sizeU(2)
40      if u==U(t)
41          s=s+1;
42      end
43  end
```

Den Suchbereich für Minima und Maxima für große Splines kann man mit folgendem Algorithmus bestimmen.

```
1   function s=suchbereich(P,U,m,p)
2   %Definiert den Bereich wo Minima und Maxima der Funktionen liegen
3   %und gibt diese aus
4   Pbea=P;
5   NUmax=[];
6   for k=1:2
7       mmax=max(Pbea);
8       hv=find(Pbea==mmax);
9       lhv=length(hv);
10      for i=1:lhv
11          if hv(i)>m-p-2
12              hv(i)=m-p-2;
13          end
14          for j=hv(i):hv(i)+p+1;
15              NUmax=[NUmax U(j)];
16          end
17          Pbea(hv(i))=NaN;
18      end
19  end

21  Pbea=P;
22  NUmin=[];
23  for k=1:2
24      mmin=min(Pbea);
25      hv=find(Pbea==mmin);
```

```
26      lhv=length(hv);
27      for i=1:lhv
28          if hv(i)>m-p-2
29              hv(i)=m-p-2;
30          end
31          for j=hv(i):hv(i)+p+1;
32              NUmin=[NUmin U(j)];
33          end
34          Pbea(hv(i))=NaN;
35      end
36  end
37  %NUmin=sort(NUmin);
38  %NUmin=unique(NUmin);
39  s=[NUmin NUmax];
40  s=sort(s);
41  s=unique(s);
```

Die Bildung der Potenzen eines Intervalls kann genau und ohne Überschätzungen mit dem Algorithmus 'factorKNR' berechnet werden.

```
1   function erg=factorKNR(x,n)
2   %Potenzieren eines Intervalls

4   a=inf(x); b=sup(x);
5   m=mod(n,2);

7   if (m==1)
8       erg=infsup(a^n,b^n);
9   else
10      if in0(0,x)
11          erg=infsup(0,max(a^n,b^n));
12      else
13          if a>=0
14              erg=infsup(a^n,b^n);
15          else
16              erg=infsup(b^n,a^n);
17          end
18      end
19  end
```

Die Division mehrerer Intervalle wird mit folgendem Algorithmus berechnet.

```
1  function erg=divideKNR(x,y)
2  %Erveiterte Intervalarthimetik
3  %Division von zwei Intervallen

5  a=inf(x); b=sup(x);
6  c=inf(y); d=sup(y);

8  lx=size(x); lx=lx(1);
9  ly=size(y); ly=ly(1);
10 if (lx==1 && ly==1)
11 if (isnan(x) || isnan(y))
12     erg=infsup(NaN,NaN);
13     return;
14 end
15 erg=divideKNRausw(a,b,c,d);
16 end

18 if (lx==1 && ly==2)
19     if (isnan(x) && ly==2)
20         erg=[infsup(NaN,NaN); infsup(NaN,NaN)];
21         return;
22     end
23     erg=[];
24     ergg=divideKNRausw(a,b,c(1),d(1));
25     erg=[erg; ergg];
26     ergg=divideKNRausw(a,b,c(2),d(2));
27     erg=[erg; ergg];
28 end

30 if (ly==1 && lx==2)
31     if (isnan(y) & ly==1)
32         erg=[infsup(NaN,NaN); infsup(NaN,NaN)];
33     end
34     erg=[];
35     ergg=divideKNRausw(a(1),b(1),c,d);
36     erg=[erg; ergg];
37     ergg=divideKNRausw(a(2),b(2),c,d);
38     erg=[erg; ergg];
39 end

41 if (ly==2 && lx==2)
42     if (isnan(y) & isnan(x))
43         erg=[infsup(NaN,NaN); infsup(NaN,NaN)];
44     end
```

```
45      erg=[];
46      ergg=divideKNRausw(a(1),b(1),c(1),d(1));
47      erg=[erg; ergg];
48      ergg=divideKNRausw(a(2),b(2),c(2),d(2));
49      erg=[erg; ergg];
50  end

52  ss=size(erg); ss=ss(1);
53  if (ss==4)
54      hv1=hull(erg(1),erg(3));
55      hv2=hull(erg(2),erg(4));
56      erg=[hv1; hv2];
57  end
```

Um die erweiterte Intervallarthimetik für die Division verwenden zu können, benötigt man den Algorithmus 'divideKNRausw'. Dies ist eigentlich die Implementation der Gleichungen aus (5.1).

```
1   function erg=divideKNRausw(a,b,c,d)
2   %Erweiterte Intervallarithmetik
3   %Division

5   if ((c<=0) && (0<=d))
6       if (c==0 && d>c)
7           if (b<0)
8               erg=infsup(-Inf,b/d);
9               return;
10          end
11          if (a>0)
12              erg=infsup(a/d,Inf);
13              return;
14          end
15      end
16      if (d==0 && d>c)
17          if (b<0)
18              erg=infsup(b/c,Inf);
19              return;
20          end
21          if (a>0)
22              erg=infsup(-Inf,a/c);
23              return;
24          end
25      end
```

```
26      if (c<0 && 0<d)
27          if (b<0)
28              erg=[infsup(-Inf,b/d);infsup(b/c,Inf)];
29              return;
30          end
31          if (a>0)
32              erg=[infsup(-Inf,a/c);infsup(a/d,Inf)];
33              return;
34          end
35      end
36      if (c==0 && d==0)
37          erg=infsup(NaN,NaN);
38      end
39      if ((a<=0) && (0<=b))
40          erg=infsup(-Inf,Inf);
41      end
42  else
43      hv1=1/d;
44      hv2=1/c;
45      erg=infsup(a,b)*infsup(hv1,hv2);
46  end
```

7.2 Weitere Interessante Algorithmen

Hier werden noch einige Algorithmen beschrieben, die man beim Arbeiten mit NURBS-Kurven benötigt. Sie haben mit der Box-Einschließung von NURBS-Kurven wenig zu tun, sind aber im Laufe der Entstehung dieser Arbeit sehr hilfreich gewesen.

Den Anfang macht der Knot Insertion Algorithmus.
Hauptsächlich fügt man weitere Knoten der NURBS-Kurve hinzu um mehr Kontrollpunkte zu erhalten, womit man mehr Kontrolle über den Verlauf der Kurve erhält. Die Theorie der 'Knot Insertion' ist in Kapitel 4 genauer angeführt.
Der hier angeführte Algorithmus entspricht unter gewissen Änderungen dem „KnotInsert“ aus [8, Kapitel 5].
Die Ausgabe besteht aus den neuen Punkten des neuen Kontrollpolygons in homogenen Koordinaten, dem neuen Knotenvektor und der Anzahl der neuen Punkte.

```
1  function [nq UQ Qw]=KnotInsert(np, p, UP, P, w, u)
2  %Knot insert
```

```
3  %np.....Anzahl der P
4  %p.....Grad
5  %UP.....Original Knotenvektor
6  %Outputs
7  %nq .....Anzahl  neuer Knoten
8  %UQ .....neuer Knotenvektor
9  %Qw .....neue Punkte in homogenen Koordinaten

11 [k, s]=FindSpanMult(np, p, u, UP);
12 mp=np+p+1;
13 nq=np+1;
14 mq=nq+p+1;
15 Pw=H(P,w);

17 for i=0:k
18      UQ(i+1)=UP(i+1);
19 end
20 UQ(k+2)=u;
21 for i=k+2:mq
22      UQ(i+1)=UP(i);
23 end
24 for i=0:k-p
25      Qw(i+1,:)=Pw(i+1,:);
26 end
27 for i=k-s:np
28      Qw(i+2,:)=Pw(i+1,:);
29 end
30 for i=0:p-s
31      Rw(i+1,:)=Pw(k-p+i+1,:);
32 end

34 L=k-p+1;
35 for i=0:p-1-s
36      alpha=(u-UP(L+i+1))/(UP(i+k+1+1)-UP(L+i+1));
37      Rw(i+1,:)=alpha*Rw(i+1+1,:)+(1-alpha)*Rw(i+1,:);
38 end
39 Qw(L+1,:)=Rw(1,:);
40 Qw(k+1-s,:)=Rw(p-s,:);

42 for i=L+1:k-s-1
43      Qw(i+1,:)=Rw(i-L+1,:);
44 end;
```

Der folgende Algorithmus, stellt eine weitere Anwendung des De Boor Verfahrens

dar.
Um die Tangente an einer NURBS-Kurve zu berechnen, muss man die Kurve nicht ableiten. Der folgende Algorithmus berechnet mithilfe von 'CurvePntByCornerCut' die erste Ableitung, ohne die Formeln aus den Kapiteln 2.2. oder 3.3. zu verwenden. Hier wurde das De Boor Verfahren umgeschrieben, sodass die Tangente im Prinzip damit berechnet wird. Die Ausgabe ist die Linearkombination zweier Punkte, die auf der Tangente liegen:

```
65 alfa=(u-U(k-p+j+i+1))/(U(i+k+2)-U(k-p+j+i+1));
66 Rw(i+1,:)=alfa*(Rw(i+2,:)-Rw(i+1,:))+Rw(i+1,:);
```

Sogar der Auswertungspunkt liegt auf der Tangente und somit kann man die Tangente an dieser Stelle bilden, indem man die Differenz aus dem Auswertungspunkt und einem der Iterationspunkte berechnet.
Die Ausgabe dieses Algorithmus besteht aus $C(u)$ und $C'(u)$

```
1  function [tv, C]=tang(n,p,P,w,U,u)
2  %[tv, C]=tangw(n,p,P,w,U,u)
3  %Berechnet Tangentenvektor an u und den Punkt C der Kurve an u
4  %mit der Cornercut Methode
5  %n....Anzahl der Punkte (0-n)
6  %p....Grad der Spline
7  %U....Knotenvektor
8  %P....Kontrollpunkte
9  %w....Gewichte
10 %u....Punkt an dem Ausgewertet wird
11 %Outputs
12 %tv...Tangentenvektor
13 %C....Punkt der Kurve

15 Pw=H(P,w); C=[];
16 %Sonderfall Endpunkte
17 if(u==U(1))
18     tv=P(2,:)-P(1,:); C=P(1,:);
19     return;
20 end
21 if(u==U(n+p+2))
22     tv=P(n+1,:)-P(n,:); C=P(n+1,:);
23     return;
24 end

26 %Linearkombinationen und Punktberechnung
27 [k, s]=FindSpanMult(n,p,u,U);
28 r=p-s; Rw=[];
```

```
29 for i=0:r
30     Rw(i+1,:)=Pw(k-p+i+1,:);
31 end

33 for j=1:r
34     for i=0:r-j
35         alfa=(u-U(k-p+j+i+1))/(U(i+k+1+1)-U(k-p+j+i+1));
36           Rw(i+1,:)=alfa*Rw(i+1+1,:)+(1-alfa)*Rw(i+1,:);
37     end;
38 end;

40 C=Hinv(Rw(1,:));
41 R=Hinv(Rw(2,:));
42 tv=R-C;
```

Oft ist es interessant, die Werte der Basisfunktionen zu berechnen. Der folgende Algorithmus (wieder aus [8] mit Änderungen übernommen) berechnet den Wert der `i`-ten Basisfunktion, an der Stelle `u`.

```
1  function Nip=OneBasisFun(p, m, U, i, u)
2  %Nip=OneBasisFun(p,m,U,i,u)
3  %Berechnet den Wert einer BasisFunktion Nip an der Stelle u
4  %p....Grad der Spline
5  %m....|U|=m+1
6  %U....KnotVektor
7  %i....Index  des Teilintervalls (beachte i\in[0,m+1])
8  %     Nummer der Basisfunktion.
9  %u....Punkt
10 %Output
11 %Nip..Wert der Basisfunktion an der Stelle u

13 if ( (i==0 && u==U(1)) || (i==m-p-1 && u==U(m+1)) )
14     Nip=1;  return;
15 end

17 if (u<U(i+1) || u >= U(i+p+1+1))
18     Nip=0;     return;
19 end

21 N=0;
22 for j=0:p
23     if (u >=U(i+j+1)&& u<U(i+j+1+1))
24         N(j+1)=1;
```

```
25      else
26          N(j+1)=0;
27      end
28  end

30  for k=1:p
31      if (N(1)==0)
32          saved=0;
33      else
34          saved=((u-U(i+1))*N(1))/(U(i+k+1)-U(i+1));
35      end
36      for j=0:p-k
37          Uleft=U(i+j+1+1);  Uright=U(i+j+k+1+1);
38          if (N(j+1+1)==0)
39              N(j+1)=saved;  saved=0;
40          else
41              temp=N(j+1+1)/(Uright-Uleft);
42              if (isnan(temp) || isinf(temp))
43                  temp=0;
44              end
45              N(j+1)=saved+(Uright-u)*temp;
46              saved=(u-Uleft)*temp;
47          end
48      end
49  end
50  Nip=N(1);
```

Im Weiteren und als Abschluss werden noch zwei Algorithmen zur graphischen Darstellung der NURBS-Kurven angeführt.

Um die Basisfunktionen einer NURBS-Kurve oder einer Spline-Kurve zu zeichnen kann man die folgende Implementation verwenden.
Gibt man dabei dem Argument `z` den Wert 1, so wird die `i`-te Basisfunktion gezeichnet. Hat `z` aber den Wert 2 so werden alle Basisfunktionen dargestellt (das Argument `i`, hat in diesem Fall keine Bedeutung).

```
1  function r=zeichneOneBasisFun(p,U,i,z)
2  %Zeichnet B-Splines Basisfunktionen
3  %z=1 Eine (die i-te) Funktion
4  %z=2 Alle Funktionen

6  a=U(1); b=U(end);
7  sizeU=size(U);
```

```
8  m=sizeU(2)-1;
9  x=a:0.005:b;
10 sizex=size(x);
11 s=sizex(2);
12 if(z==1)
13     for t=1:s
14         Nip(t)=OneBasisFun(p, m, U, i, x(t));
15     end
16     r=Nip;
17 plot(x,Nip);
18 else
19     n=m-p-1;
20     for k=0:n
21         for t=1:s
22             Nip(k+1,t)=OneBasisFun(p, m, U, k, x(t));
23         end
24     end
25     r=Nip;
26     plot(x,Nip)
27 end
```

Die NURBS-Kurven selbst kann man mit folgender Implementation graphisch darstellen. Die Genauigkeit ist dabei ein Tausendstel. Verändert man in der Zeile 7, den Unterteilungswert, so kann man die Genauigkeit der graphischen Darstellung frei wählen.

```
1  function zeichneNURBSKurve(n,p,U,P,w,sp)
2  %zeichnet NURBS-Kurven
3  %sp.....Farbe

5  m=n+p+1;
6  a=U(1); b=U(end);
7  x=a:0.001:b;
8  sizex=size(x);
9  s=sizex(2);
10 for t=1:s
11     xt=x(t);
12     Nip(t,:)=CurvePntByCornerCut(n, p, U, P,w, x(t));
13 end
14 sizeP=size(P); d=sizeP(2);
15 if (d==1)
16     plot(x,Nip,sp)
17 end
```

```
18 if (d==2);
19     plot(Nip(:,1),Nip(:,2),sp);
20 end
21 if (d==3);
22     plot3(Nip(:,1),Nip(:,2),Nip(:,3),sp);
23 grid on
24 end
```

Listings

Literaturverzeichnis

[1] R.W. Freund and R.H.W. Hoppe. *Stoer / Bulirsch: Numerische Mathematik 1*. Springer-Verlag Berlin Heidelberg, 2007.

[2] E. Hansen. *Global optimization using interval analysis*. Marcel Dekker, New York, 1992.

[3] G.I. Hargreaves. Interval analysis in matlab. *Numerical Analysis Report*, 416, 2002.

[4] F. Jarre. *Optimierung*. Springer, 2004.

[5] M. Knorrenschild. *Numerische Mathematik: Eine beispielorientierte Einführung*. Hanser Verlag, 2008.

[6] Ramon E. Moore, R. Baker Kearfott, and Michael J. Cloud. *Introduction to Interval Analysis*. siam, 2009.

[7] A. Neumaier. Computer graphics, linear interpolation, and nonstandard intervals.

[8] L.A. Piegl and W. Tiller. *The NURBS book*. Springer, 1997.

[9] R. Plato. *Numerische Mathematik kompakt: Grundlagenwissen für Studium und Praxis*. Vieweg Friedr. und Sohn Ver, 2006.

[10] J.J. Risler. *Mathematical methods for CAD*. Cambridge Univ Pr, 1992.

[11] Siegfried M. Rump. Intlab-interval laboratory. www.ti3.tu-harburg.de/rump/intlab/.

[12] J. Stoer and R. Bulirsch. *Numerische Mathematik 2*. Springer, 2005.

[13] Wayne Tiller. Knot-removal algorithms for nurbs curves and surfaces. *CAD*, 24, 1992.

Lebenslauf

Name:	Amra Smajic
Geburtstdaten:	04.03.1984, Livno, Bosnien und Herzegowina
Familienstand:	ledig

Schulbildung:

1990-1995	Volksschule in Bosnien und Herzegowina
1995-1999	Hauptschule in Gaming, Niederösterreich
1999-2003	Oberstufenralgymnasium mit Informatik in Scheibbs, Niederösterreich

Studium:

2003-2009	Diplomstudium Mathematik, Universität Wien

Berufliche Laufbahn:

Juli, August 2007	Praktikum bei der Firma Worthington Cylinders in Kienberg (NÖ), Schuluingsseminare in Novell
2007-2009	Vorsitz der Fakultätsvertretung für Mathematik an der Uni Wien
SS06, SS07, WS07, WS08	Tutorin an der Fakultät für Mathematik