Übungen Approximation

Gerhard Opfer SS 1999

Bogen 12: Übungen am 6. Juli 1999

Lösungshinweise

Präsenzaufgaben:

Aufgabe P34. Seien $v_0, v_1, \ldots, v_n$ und $w_0, w_1, \ldots, w_n$ zwei Systeme von $n+1$ Polynomen in Π_n, und bilden die w_j, $j = 0, 1, \ldots, n$ eine Basis von Π_n. Wir fassen die v_j zu einer Spalte $\mathbf{v}$ und die w_j zu einer Spalte $\mathbf{w}$ zusammen. Wegen der linearen Unabhängigkeit der w_j hat dann jedes v_j eine eindeutige Darstellung als Linearkombination der w_j, es gibt also eine Matrix $\mathbf{A}$ mit $\mathbf{v} = \mathbf{A}\mathbf{w}$. Sei $\mathbf{a} \in \mathbb{R}^{n+1}$ und seien auch die in $\mathbf{v}$ versammelten Polynome linear unabhängig, dann ist

$$\mathbf{a}^{\mathrm{T}}\mathbf{v} = \mathbf{a}^{\mathrm{T}}\mathbf{A}\mathbf{w} = (\mathbf{A}^{\mathrm{T}}\mathbf{a})^{\mathrm{T}}\mathbf{w} = 0 \Rightarrow (\mathbf{A}^{\mathrm{T}}\mathbf{a})^{\mathrm{T}} = \mathbf{0} \text{ und } \mathbf{a} = \mathbf{0},$$

die Matrix $\mathbf{A}$ ist also nicht singulär. Ist umgekehrt die Matrix $\mathbf{A}$ nicht singulär, so sind die Polynome in $\mathbf{v}$ (im wesentlichen nach derselben Rechnung) linear unabhängig, und man erhält die inverse Darstellung $\mathbf{w} = \mathbf{A}^{-1}\mathbf{v}$. Sei $w_j(t) := t^j$ die Monombasis. Dann ist $v_j = \sum_{k=0}^{n} a_{jk} t^k$, mit bestimmten Koeffizienten a_{jk}, die wir wie oben zu $\mathbf{A} = (a_{jk})$ zusammenfassen, also $\mathbf{v} = \mathbf{A}\mathbf{w}$. Ist $w_j = t^j(1-t)^{n-j}$ das System der Bernsteinpolynome ohne den Faktor $\binom{n}{j}$, so kann man die Matrix $\mathbf{A}$ einfach bestimmen (es ist immer $0 \le j, k \le n$):

(L12.1)

$$w_j(t) = t^j \sum_{k=0}^{n-j} \binom{n-j}{k} (-t)^k = \sum_{k=j}^{n} (-1)^{k-j} \binom{n-j}{k-j} t^k \Rightarrow a_{jk} = \begin{cases} (-1)^{k-j}\binom{n-j}{k-j} & \text{für } k = j, j+1, \ldots, n, \\ 0 & \text{für } k < j. \end{cases}$$

Die Matrix $\mathbf{A}$ ist eine obere Dreiecksmatrix mit nur ganzzahligen Einträgen und alle Diagonalelemente sind Eins. Die inverse Matrix existiert also, ist ebenfalls eine obere Dreiecksmatrix auch mit nur ganzzahligen Einträgen. Damit ist nachgewiesen, daß die Bernsteinpolynome eine Basis bilden. Eine Darstellung der Monome durch Bernsteinpolynome erhält man also durch Inversion von $\mathbf{A}$. Für $n = 3$ erhält man speziell

$$\mathbf{A} = \begin{pmatrix} 1 & -3 & 3 & -1 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \Rightarrow \mathbf{B} := \mathbf{A}^{-1} = \begin{pmatrix} 1 & 3 & 3 & 1 \\ 0 & 1 & 2 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Also $t^0 = 1 = b_0^3 + 3b_1^3 + 3b_2^3 + b_3^3$; $t^1 = t = b_1^3 + 2b_2^3 + b_3^3$; $t^2 = b_2^3 + b_3^3$; $t^3 = b_3^3$.

Aufgabe P35. Betrachten wir nur das *Knotenpolynom* $K_j := \prod_{\substack{k=0\\k\neq j}}^{n} (t - t_k)$, so ist $K_j = \sum_{k=0}^{n} a_{jk} t^k$ und die a_{jk} sind die sogenannten *elementar symmetrischen Funktionen*, d. h.

$$\text{(L12.2)} \qquad a_{jn} = 1,\ a_{j,n-1} = -\sum_{\substack{k=0\\k\neq j}}^{n} t_k,\ a_{j,n-2} = \sum_{\substack{k_1<k_2\\k_1\neq j, k_2\neq j}}^{n} t_{k_1}t_{k_2}, \ldots, a_{j0} = (-1)^n t_0 t_1 \cdots t_{j-1} t_{j+1} \cdots t_n.$$

Für $n = 3$ und $t_k = k$, $k = 0, 1, 2, 3$ erhalten wir

$$\mathbf{A} = \begin{pmatrix} -6 & 11 & -6 & 1 \\ 0 & 6 & -5 & 1 \\ 0 & 3 & -4 & 1 \\ 0 & 2 & -3 & 1 \end{pmatrix} \Rightarrow \mathbf{B} := \mathbf{A}^{-1} = \begin{pmatrix} -1/6 & 1/2 & -1/2 & 1/6 \\ 0 & 1/2 & -1 & 1/2 \\ 0 & 1/2 & -2 & 3/2 \\ 0 & 1/2 & -4 & 9/2 \end{pmatrix}.$$

Also $1 = -K_0/6 + K_1/2 - K_2/2 + K_3/6$, $t = K_1/2 - K_2 + K_3/2$, $t^2 = K_1/2 - 2K_2 + 3K_3/2$, $t^3 = K_1/2 - 4K_2 + 9K_3/2$. Aus $\sum_{j=0}^{n} a_j \ell_j(t) = 0$ für alle t folgt durch Einsetzen aller Knoten t_k sofort $a_0 = a_1 = \cdots = a_n = 0$, die Lagrange-Koeffizienten sind also linear unabhängig, die obige Matrix (L12.2) $\mathbf{A}$ also nicht singulär.

Aufgabe P36. Die diskutierten Bézier-Kurven haben die Darstellung

$$\mathbf{b}^n(t) := \sum_{j=0}^{n} \mathbf{b}_j b_j^n(t) \text{ mit } b_j^n(t) := \binom{n}{j} t^j (1-t)^{n-j}. \tag{L12.3}$$

Insbesondere gilt

$$b_j^n(0) = b_j^n(1) = 0 \text{ für alle } j = 1, 2, \ldots, n-1 \text{ und } b_0^n(0) = 1, b_n^n(0) = 0, b_0^n(1) = 0, b_n^n(1) = 1.$$

Daraus folgt, wie behauptet

$$\mathbf{b}^n(0) = \mathbf{b}_0, \quad \mathbf{b}^n(1) = \mathbf{b}_n.$$

Daß die Kurve nicht durch die anderen Kontrollpunkte geht, sieht man am Beispiel der Abbildung L12.4.

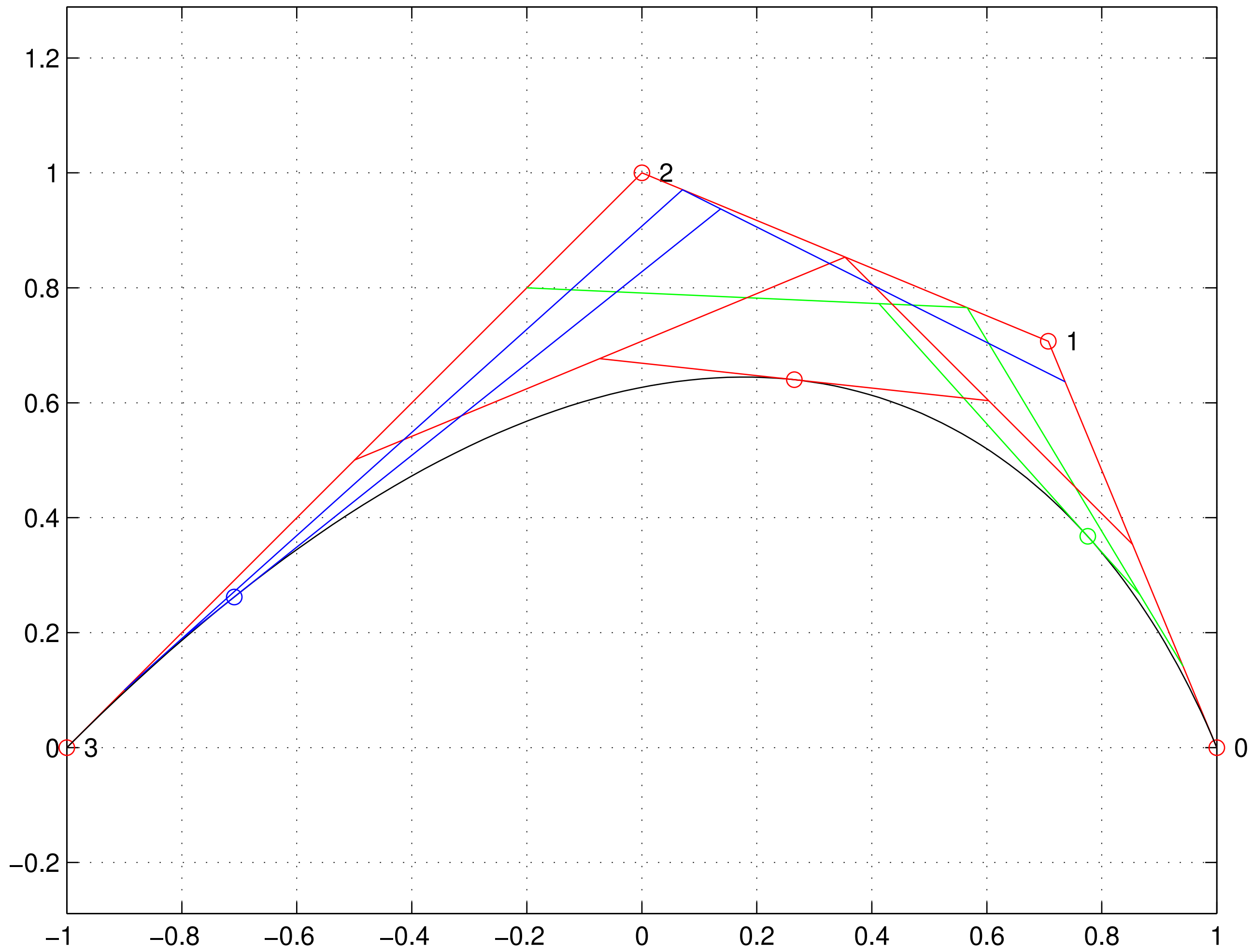

Abbildung L12.4. Anwendung des de Casteljau-Algorithmus zu $t = 0.2,\ 0.5,\ 0.9$

Hausaufgaben:

Aufgabe 23. Aus (L12.3) folgt für die Ableitung der Bézier-Kurven (vgl. FARIN [1994, S. 38])*)

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{b}^n(t) = n \sum_{j=0}^{n-1} (\mathbf{b}_{j+1} - \mathbf{b}_j) b_j^{n-1}(t). \tag{L12.5}$$

*) GERALD FARIN, Kurven und Flächen im Computer Aided Geometric Design, 2. Auflage, Vieweg, Braunschweig, Wiesbaden, 1994, 365 S.

Daraus folgt sofort

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{b}^n(0) = n(\mathbf{b}_1 - \mathbf{b}_0), \quad \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{b}^n(1) = n(\mathbf{b}_n - \mathbf{b}_{n-1}),$$

die Richtungen am Anfang und am Ende stimmen also mit den entsprechenden Richtungen des Kontrollpolygons überein.

Aufgabe 24. Das Bild ist für $t = 0.2, 0.5, 0.9$ in Abbildung L12.4 wiedergegeben, die dazugehörigen Programme in Programm L12.6 und L12.7.

Programm L12.6. de Casteljau-Algorithmus

```
%(1) Die vorgegebenen Kontrollpunkte werden gezeichnet,
%(2) numeriert,
%(3) verbunden (Kontrollpolygon),
%(4) die Bezier-Kurve wird gezeichnet, (spezielle Prozedur wird verwendet)
%(5) nach dem de Casteljau-Algorithmus werden einige Kurvenpunkte gezeichnet.
%Programm: G. Opfer, Juli 1999

Kontrollpunkte=[cos(0),sin(0);cos(pi/4),sin(pi/4);cos(pi/2),sin(pi/2);...
               cos(pi),sin(pi)];
%(1) ------------------------------------------------------------------------
zz=Kontrollpunkte(:,1)'+i*Kontrollpunkte(:,2)';
plot(zz,'ro'); hold on     %gibt Kontrollpunkte in rot
%(2) ------------------------------------------------------------------------
for j=1:length(zz) %ergibt Numerierung in schwarz
  text(real(zz(j))+0.03,imag(zz(j)),[int2str(j-1)]);
end; %for j
%(3) ------------------------------------------------------------------------
plot(zz,'r');     %gibt Kontrollpolygon auch in rot
axis equal
grid on
%(4) ------------------------------------------------------------------------
t=linspace(0,1);
[Kurve]=bezier(Kontrollpunkte,t);
plot(Kurve(1,:),Kurve(2,:),'k'); %in schwarz
%(5) ------------------------------------------------------------------------
%T=[0.2 1/3 0.5 0.75 0.9];  %de Casteljau-Punkte (zum Zeichnen zu viele)
T=[0.2 0.5 0.9];            %de Casteljau-Punkte
[m,n]=size(Kontrollpunkte); %n=2 wird nicht gebraucht
%m = Anzahl der Kontrollpunkte
k=0;
for t = T
  mm=m;
  controlpoints=Kontrollpunkte;
  k=k+1;
  while mm>1
    controlpoints(1:mm-1,:)=...
                     (1-t)*controlpoints(1:mm-1,:)+t*controlpoints(2:mm,:);
    controlpoints(mm,:)=[]; %am Schlu\ss\ enth\"alt die Matrix nur eine Zeile
    z=controlpoints(:,1)'+i*controlpoints(:,2)';
    Farbe='grbyk'; %f\"ur jedes t eine Farbe: rot, gr\"un, blau, gelb, schwarz
    Farbe2=['go';'ro';'bo';'yo';'ko']; %farbiger kleiner Kreis
    if mm>2
      plot(z,Farbe(k));
      else
      plot(z,Farbe2(k,:)); %der eigentliche Kurvenpunkt
    end; %if
    mm=mm-1;
  end; %while
end; %for t
hold off
%print -dpsc deC.ps
```

Programm L12.7. Bézier-Kurven

```
%Aus gegebenen Kontrollpunkten (b(j,1),b(j,2)), j=1,2,...,m
%wird die entsprechende Bezier-Kurve vom Grad m-1 gezeichnet.
%Es wird die Darstellung mit Bernsteinpolynomen benutzt.
%Die Form ist function [Kurve]=bezier(Kontrollpunkte,t);
%Das Ergebnis ist eine Kurve, die in der Form plot(t,Kurve)
%geplottet werden kann. Will man sie abspeichern, ist in der
%Kommandazeile ein Befehl der Form
%   "print -dpsc bildkurve.ps"
%(ohne "  ") zu schreiben. Es werden zwei Unterprogramme benutzt:
%function [b]=bernstein(p,t); %Bernsteinpolynome vom Grad p an den Stellen t
%function [b]=bin(n);         %alle Binomialkoeffizienten n \"uber j, j=0,1,...,n
%Die Programme sind einfach zu durchschauen, aber nicht besonders effektiv.
```

```
%Besser w\"aren rekursive Formen. Programm: G. Opfer, Juli 1999

function [Kurve]=bezier(Kontrollpunkte,t);
[m,n]=size(Kontrollpunkte);
if n~=2
  error('Kontrollpunkte haben nicht genau zwei Komponenten');
end; %if
[b]=bernstein(m-1,t);
M=1:m;
Kurve=zeros(2,length(t));
for z=1:2
  for mm=M
    Kurve(z,:)=Kurve(z,:)+Kontrollpunkte(mm,z)'.*b(mm,:);
  end; %for mm
end; %for z;
%---------------------------------------------------------------------------
%Es werden alle Bernsteinpolynome b vom Grad p, ausgewertet an den
%Stellen t berechnet. b ist eine Matrix b(j,tau), j=1,2,...,p+1,
%tau = length(t); intern wird "bin" ("function [b]=bin(n);")
%zur Berechnung aller Binomialkoeffizienten benutzt.
%function [b]=bernstein(p,t);
function [b]=bernstein(p,t);
tau=length(t);
Tau=1:tau;
P=1:p+1;
b=zeros(p+1,tau);
bi=bin(p);
for pp=P
  b(pp,:)=bi(pp)*((1-t).^(p-pp+1).*t.^(pp-1));
end; %for
%---------------------------------------------------------------------------
%In b stehen alle n+1 Binomialkoeffizienten zum Grad n zeilenweise
%function [b]=bin(n);
function b=bin(n);
J=1:n+1;
for j=J
  b(j)=prod((n:-1:n-j+2)./(j-1:-1:1));
end; %for
```

Die Aufgaben und mit einiger Verzögerung auch die Lösungen können von den web-Seiten kopiert werden:

http://www.math.uni-hamburg.de/home/opfer/aufgaben99A.html