Prof. Dr. Stefan Funken
Dipl.-Math. oec. Klaus Stolle
Institut für Numerische Mathematik
Universität Ulm

WiSe 2016/2017
Blatt 6
23.11.2016

Angewandte Numerik 2

Besprechung in den Tutorien in der Woche vom 28.11.2016 bis 02.12.2016

Für dieses Übungsblatt gibt es 14 Theorie- und 23 Matlab-Punkte, sowie 4 Theorie- und 15 Matlab-Zusatzpunkte. Punkte, die mit einem * gekennzeichnet sind, sind Zusatzpunkte.
Die 60-Prozent-Grenzen liegen aktuell (inklusive Blatt 6) bei 49,8 Theoriepunkten und 61,8 Matlabpunkten.

Aufgabe 19 *(Bernstein-Polynome)* (1T+2T+2T+2T* Punkte)

Bestimmen Sie

a) das konstante Bernstein-Polynom (also das Bernstein-Polynom vom Grad 0)

sowie mit Hilfe der Rekursionsformel

b) die linearen Bernstein-Polynome (also die Bernstein-Polynome vom Grad 1),

b) die quadratischen Bernstein-Polynome und

b) die kubischen Bernstein-Polynome.

Aufgabe 20 *(Darstellung der Bernstein-Polynome)* (8M* Punkte)

Schreiben Sie ein Matlabskript `bernsteinPolynome`, mit dem Sie die Bernstein-Polynome vom Grad 0 bis zum Grad n plotten können. Wählen Sie eine effiziente Implementierung. Testen Sie Ihr Matlabskript mit $n = 8$.

Aufgabe 21 *(Programmieraufgabe: Bézier-Kurven im $\mathbb{R}^2$)* (5M+5M+5M+1M+4T+2T Punkte)

a) Schreiben Sie eine Matlabfunktion `C = bezier(P,t)`, die die Bézier-Kurve zu den Kontrollpunkten `P` an den Stellen `t` auswertet. `P` ist dabei der Vektor $(\boldsymbol{P}_0, \ldots, \boldsymbol{P}_n)$ der $n+1$ Kontrollpunkte $\boldsymbol{P}_j \in \mathbb{R}^2$, also eine Matrix $\in \mathbb{R}^{2\times(n+1)}$. `t` ist ein Vektor $(t_1, \ldots, t_m)$. Ihre Funktion soll die Werte $\boldsymbol{C}(t_j) \in \mathbb{R}^2$, $j = 1, \ldots, m$ der Bézier-Kurve in der Matrix $\texttt{C} = (\boldsymbol{C}(t_1), \ldots, \boldsymbol{C}(t_m)) \in \mathbb{R}^{2\times m}$ zurück geben.

b) Schreiben Sie ein Matlabskript `bezierKurve`, bei dessen Aufruf der Benutzer aufgefordert wird, die Kontrollpunkte $\boldsymbol{P}_0, \ldots, \boldsymbol{P}_n$ zu markieren. Bei der Eingabe soll Ihr Skript das zugehörige Kontrollpolygon zeichnen. Nach Eingabeende soll die Bézier-Kurve zu den eingegebenen Kontrollpunkten geplottet werden.
Hinweis: Sie können sich beim Einlesen der Kontrollpunkte am Matlabskript `drawSpline` von Aufgabe 5 orientieren.

c) Schreiben Sie eine Matlabfunktion `C0 = deCasteljauVisual(P,t0)`, die den Punkt `C0` der Bézier-Kurve zu den Kontrollpunkten `P` (wie in Aufgabenteil a)) an der Stelle `t0` $\in [0, 1]$ mit dem De-Casteljau-

Algorithmus berechnet. Ihre Funktion soll den De-Casteljau-Algorithmus verdeutlichen und dazu die Polygonzüge durch die Punkte $\boldsymbol{P}_{k,j}(t_0)$ mit $k = 1, \ldots, n$ und $j = 0, \ldots, n-k$ zeichnen sowie die Ecken $\boldsymbol{P}_{k,j}(t_0)$ durch kleine Kreise markieren.

d) Ergänzen Sie Ihr Matlabskript `bezierKurve`, so dass Sie Ihre Matlabfunktion `deCasteljauVisual` testen können.

e) Erläutern Sie, wie man mit Hilfe des De-Casteljau-Algorithmus eine Bézier-Kurve berechnen kann. Wählen Sie für Ihre Erläuterungen ein einfaches Beispiel mit 4 Kontrollpunkten $P_j \in \mathbb{R}^2,\ (j = 0, ..., 3)$.

f) Haben Sie Ihre Matlabfunktion `bezier` aus Aufgabenteil a) effizient implementiert? Begründen Sie Ihre Antwort.

Aufgabe 22 *(Programmieraufgabe: Approximation von Funktionen mit Bernsteinpolynomen)*
(3T+5M+1M+1M+2M*+2M*+(2T*+3M*) Punkte)

In dieser Aufgabe wollen wir numerisch verifizieren, dass auf einem Intervall $[0, 1] \subset \mathbb{R}$ stetige Funktionen $f \in C[0, 1]$ sich durch eine Linearkombination der Bernsteinpolynome approximieren lassen.
Dazu definieren wir für $t \in [0, 1]$ das n-te ($n \in \mathbb{N}$) Approximations-Polynom zu f durch

$$F_n(f; t) = \sum_{i=0}^{n} f\left(\frac{i}{n}\right) B_i^n(t),$$

wobei die $B_i^n(t),\ (i = 0, \ldots, n)$ die Bernstein-Polynome sind. Für große n gilt für alle $t \in [0, 1]$: $f(t) \approx F_n(f; t)$.

a) Betrachten Sie zunächst die Funktion $f : [0, 2\pi] \to \mathbb{R}, f(x) = \sin(x)$. Überlegen Sie sich, wie Sie das Intervall $[a, b] = [0, 2\pi]$ auf das Intervall $[0, 1]$ transformieren können. Wie können Sie $F_n(f; t)$ auf das Intervall $[a, b]$ verallgemeinern?

b) Berechnen Sie für $n \in \{2^1, \ldots, 2^{10}\}$ und für $t_1, \ldots, t_m \in [0, 2\pi]$ mit $m \gg n$ (beispielsweise $m = 10\,n$) $F_n(f; t_1), \ldots, F_n(f; t_m)$ und den Näherungswert $e_f(n) = \max_{k=1,\ldots,m} |f(t_k) - F_n(f; t_k)|$ für den Approximationsfehler.

Achtung: Für große n dauern die Berechnungen einige Minuten. Testen Sie Ihre Implementierung daher zuerst für kleine Werte von n.

c) Zeichnen Sie die Funktion $f(x) = \sin(x)$ und ihre Approximation $F_n(f; x)$ für $n = 2^{10}$ im Intervall $[0, 2\pi]$ in ein Schaubild.

d) Plotten Sie für die in Aufgabenteil b) gegebenen Werte für n den Approximationsfehler $e(n)$ über n in ein neues Schaubild mit logarithmischen Achsen.

e) Betrachten Sie nun die Funktion $g(x) = |x|$ im Intervall $[-1, 1]$. Berechnen Sie analog die Näherungswerte $F_n(g; t_k)$ (mit $t_k \in [-1, 1]$) sowie die Approximationsfehler $e_g(n)$. Zeichnen Sie wiederum die Funktion g und ihre Approximation $B_n(g; \cdot)$ sowie den Approximationsfehler über n.

f) Untersuchen Sie nun auch Runges Beispielfunktion $h(x) = \frac{1}{1+x^2}$ auf dem Intervall $[-5, 5]$. Warum kann diese Funktion durch $F_n(h; \cdot)$ ohne Oszillationen approximiert werden?

g) Überlegen Sie sich, ob und wie Sie den Algorithmus von de Casteljau zur Berechnung der Näherungswerte $F_n(f; t_k)$ einsetzen können. Verifizieren Sie Ihre Überlegungen mit Matlab. Welche Implementierung braucht weniger Rechenzeit, ist also effizienter?

Hinweise:
Die Programmieraufgaben sind in Matlab zu erstellen. Der Source Code muss strukturiert und dokumentiert sein. Senden Sie spätestens am Tag vor Ihrem Tutorium alle Matlab-Files und alle Ergebnisse in einer E-mail mit dem Betreff **Loesung-Blatt06** an **angewandte.numerik@uni-ulm.de**