Familienname:

Vorname:

Matrikelnummer:

Aufgabe 1 (8 Punkte):
Aufgabe 2 (2 Punkte):
Aufgabe 3 (10 Punkte):
Aufgabe 4 (10 Punkte):

Gesamtpunktzahl:

Schriftlicher Nachtest zu Matlab (90 Minuten)
VU Einführung ins Programmieren für TM (WS 2009/10)

01. März 2010

Aufgabenstellung. Das Newton-Verfahren erlaubt die effiziente numerische Berechnung von Nullstellen von Polynomen $p(x) = \sum_{j=0}^{n} a_{j+1} x^j$. Dabei bezeichnet man $n \in \mathbb{N}$ als Grad von p, sofern $a_{n+1} \neq 0$ gilt. Im Folgenden sollen Polynome in Form des Koeffizientenvektors $a = (a_1, \ldots, a_{n+1}) \in \mathbb{R}^{n+1}$ gespeichert werden. Der Test gliedert sich wie folgt:

- Berechnung des Funktionswerts $p(x)$ eines Polynoms mittels Horner-Schema.
- Berechnung der Ableitungen eines Polynoms.
- Realisierung des Newton-Verfahrens $x_{\ell+1} = x_\ell - p(x_\ell)/p'(x_\ell)$ zur Approximation einer Nullstelle von p.

Aufgabe 1 (8 Punkte). Schreiben Sie eine Funktion `evalPolynomial`, die ein Polynom p an einer gegebenen Stelle x *möglichst effizient* auswertet. Dabei sollen lediglich die skalare Addition und Multiplikation, nicht aber der Dachoperator `x^j` verwendet werden. — Naive Realisierung führt auf quadratischen Aufwand in Bezug auf $n = \mathrm{Grad}(p)$, d.h. man benötigt $\mathcal{O}(n^2)$ arithmetische Operationen (zwei geschachtelte Schleifen!). Alternativ kann man durch Herausheben (Ausklammern) von x in der Darstellung $p(x) = \sum_{j=0}^{n} a_{j+1} x^j$ einen Algorithmus entwickeln, der nur linearen Aufwand hat, d.h. man benötigt nur $\mathcal{O}(n)$ arithmetische Operationen (eine Schleife!). Man bezeichnet diesen Algorithmus als *Horner-Schema*. — Machen Sie sich das Horner-Schema zunächst für $n = 2, 3, 4$ klar!

Lösung zu Aufgabe 1.

Aufgabe 2 (2 Punkte). Realisieren Sie die Funktion `evalPolynomial` in möglichst wenig Zeilen unter Verwendung aller Stärken der MATLAB-Arithmetik, d.h. Sie dürfen alles nutzen, was MATLAB Ihnen bietet.

Lösung zu Aufgabe 2.

Aufgabe 3 (10 Punkte). Die k-te Ableitung $p^{(k)}$ eines Polynoms $p = \sum_{j=0}^{n} a_{j+1} x^j$ ist wieder ein Polynom. Man schreibe eine Funktion `derivativePolynomial`, die zu gegebenen k und p die Ableitung $p^{(k)}$ (bzw. dessen Koeffizientenvektor) berechnet. — Die Funktion soll möglichst ohne Zusatzspeicher auskommen. Ferner soll der Koeffizientenvektor von $p^{(k)}$ minimale Länge haben. — Überlegen Sie sich zunächst, welchen Grad $p^{(k)}$ hat und wie sich die Koeffizienten berechnen.

Lösung zu Aufgabe 3.

Aufgabe 4 (10 Punkte). Gegeben seien ein Startwert $x_0 \in \mathbb{R}$, eine Toleranz $\tau > 0$ und der Koeffizientenvektor eines Polynoms p vom Grad n. Für einen hinreichend großen Startwert x_0 konvergiert dann die induktiv definierte Newton-Folge

$$x_{\ell+1} := x_\ell - \frac{p(x_\ell)}{p'(x_\ell)} \quad \text{für } \ell \in \mathbb{N}$$

gegen die größte Nullstelle x^* von p. Schreiben Sie eine Funktion `rootPolynomial`, die die Folgenglieder x_ℓ berechnet, bis sowohl

$$|p(x_\ell)| \leq \tau$$

als auch

$$|x_\ell - x_{\ell-1}| \leq \begin{cases} \tau & \text{für } |x_\ell| \leq \tau \\ \tau\,|x_\ell| & \text{sonst} \end{cases}$$

gelten. In diesem Fall werde x_ℓ als Approximation von x^* zurückgegeben. — Bei der Realisierung verwende man die Funktionen aus Aufgabe 1 und 3. Achten Sie auf die Effizienz Ihrer Funktion, indem Sie unnötige Funktionsaufrufe vermeiden. Speichern Sie *nicht* die ganze Folge $x_0, \ldots, x_\ell$, der Iterierten, sondern lediglich jeweils die beiden letzten Werte $x_{\ell-1}$ und x_ℓ.

Lösung zu Aufgabe 4.