Prof. Dr. Stefan Funken
M.Sc. Andreas Bantle
Dipl.-Math. oec. Klaus Stolle

Universität Ulm
Institut für Numerische Mathematik
Sommersemester 2015

Numerische Analysis - Matlab-Blatt 3
Lösung

(Besprechung in den MATLAB-Tutorien in KW 21/22)

Hinweise:
Siehe MATLAB-Blatt 1/2.

Aufgabe 4 *(Horner's Scheme)* (10+5 Punkte)

(i) Write a MATLAB function

`[fx,dfx] = hornerNewton(a,x,t)`,

which evaluates the polynomial

$$\begin{aligned} p(t) &= a_0 + a_1(t-x_0) + a_2(t-x_0)(t-x_1) + \ldots + a_{n+1}(t-x_0)(t-x_1)\cdot\ldots\cdot(t-x_n) \\ &= a_0 + (t-x_0)\big(a_1 + (t-x_1)(a_2 + \ldots(a_n + (t-x_n)a_{n+1})\ \ldots\ \big) \end{aligned}$$

in Newton representation and its derivative with the Horner scheme. The input parameters are the coefficient vector `a` $:= (a_0, \ldots, a_{n+1})$, the interpolation points `x` $:= (x_0, \ldots, x_n)$, and the evaluation points `t`.

```
function [fx,dfx] = hornerNewton(a,x,x0)
% evaluate Newton polynomial and its derivative
% using Horner's scheme
% p(x0) = a(1) + a(2)*(x0-x(1)) + a(3)*(x0-x(1))*(x0-x(2)) + ...
%       = a(1) + ( (x0-x(1)) * ( a(2) + (x0-x(2)) * ( a(3) ....
fx = a(end);  dfx = 0;
for k=length(a)-1:-1:1
  dfx = fx + (x0-x(k)).*dfx;
  fx =  a(k) + (x0-x(k)).*fx;
end
```

(ii) Write a script `testHorner.m`, which evaluates the polynomial

$$p(t) = 2 + (t-1) + 3(t-1)(t+1) - 4(t-1)(t+1)(t-2)$$

and its derivative $p'(t)$ at $t = 3$ and $t = 4$ with the function `hornerNewton.m`. Compare the results with the exact values.

```
clear all;
close all;
clc

a = [2 1 3 -4];
x = [1 -1 2];

%% example 1
x0 = 3
[fx,dfx] = hornerNewton(a,x0,x)
```

```
11
12 %% example 2
13 x0 = 4
14 [fx,dfx] = hornerNewton(a,x0,x)
```

Erläuterungen:

1. *Die Schleife der HornerNewton-Methode fängt in der innersten Klammer des Horner Schemas an endet außen*
2. *Durch die Notation*

   $$p_{n+1}(t) = a_{n+1} \qquad p_k(t) = a_k + (t - x_k)p_{k+1}(t) \qquad \Rightarrow p_0(t) = p(t),$$

   kann man die gesuchten Ableitungen iterativ mit der Produktregel errechen,

   $$p'_k(t) = p_{k+1}(t) + (t - x_k)p'_{k+1}(t)$$

3. *Da die obige Gleichung den Wert $p_{k+1}(t)$ in jedem durchlauf benötigt ist es effizient die Auswertung der Funktion und ihrer Ableitung in einer Schleife zu errechnen, man hat also linearen Aufwand $\mathcal{O}(n)$.*

Aufgabe 5 *(Wahl der Interpolationsknoten)* (5+10 Punkte)

Wir wollen mit dieser Aufgabe den Einfluss der Wahl der Interpolationsknoten auf die Güte der Approximation einer Funktion durch ein Interpolationspolynom betrachten.

(i) Schreiben Sie eine Funktion

`c = coeffNewton(x,fx)`,

die zu den Stützstellen `x` = $(x_0, ..., x_n)$ und den Funktionswerten `fx` = $(f(x_0), ..., f(x_n))$ die Koeffizienten des Newton'sches Interpolationpolynoms mittels dividierte Differenzen berechnet und im Vektor `c` zurückliefert.

```
function fx = coeffNewton(x,fx)

% use divided differences to calculate Newton's interpolation polynomial
for k = 2:length(fx)
    for j = length(fx):-1:k
        fx(j) = (fx(j)-fx(j-1))/(x(j)-x(j-k+1));
    end
end
```

(ii) Schreiben Sie ein Skript `main.m`, welches die Funktion $f(x) = \frac{1}{x^2+1}$ auf dem Intervall $[-3, 3]$ interpoliert. Verwenden Sie hierzu verschiedene Typen von Stützstellen:

- äquidistante Stützstellen
- Nullstellen der Tschebyscheff-Polynome

$$x_k = 3 \cdot \cos\left(\frac{2k-1}{2n}\,\pi\right), \quad k = 1, ..., n.$$

- Nullstellen der Legendre Polynome
  (Verwenden Sie hierzu die MATLAB Funktion `x = rootsLegendre(n)`, welches die Nullstellen $x_1, ..., x_n$ berechnet und auf der Homepage zum Download bereit steht.)

Sortieren Sie anschließend die Nullstellen nach der Leja-Sortierung (Aufruf: `x = leja(x)`).
Zeichnen Sie das Interpolationspolynom $P_n[f]$ $(n \in \{10, 30, 1000\})$ für alle Typen von Stützstellen sowie die exakte Funktion f in ein Schaubild (Verwenden Sie hierzu die Funktionen `hornerNewton.m` und `coeffNewton.m`). Achten Sie auf die Achsenbeschriftungen und fügen Sie eine Legende ein.
Was fällt Ihnen auf?

```
clear all;
close all;
clc;

f = @(x) 1./(1+x.^2);
% f2 = @(x) atan(x);
%f2 = @(x) abs(x);

n=100;
t = 3*linspace(-1,1,1000);

%*** equidistant
xE = linspace(-3,3,n);
xE = leja(xE);
aE = coeffNewton(xE,f(xE));

%*** Tschebyscheff zeros
xT = 3*cos((2*(1:n)-1)/(2*n)*pi);
xT = leja(xT);
aT = coeffNewton(xT,f(xT));

%*** Legendre zeros
xL = rootsLegendre(n);
xL = leja(xL);
aL = coeffNewton(xL,f(xL));

```

```
%*** plot solution
fE = hornerNewton(aE,xE,t);
fT = hornerNewton(aT,xT,t);
fL = hornerNewton(aL,xL,t);

plot(t,fE,'-r',t,fT,'-b',t,fL,'-m', t,f(t),'--c')
legend('Equidistant','Tschebysheff','Legendre')
```

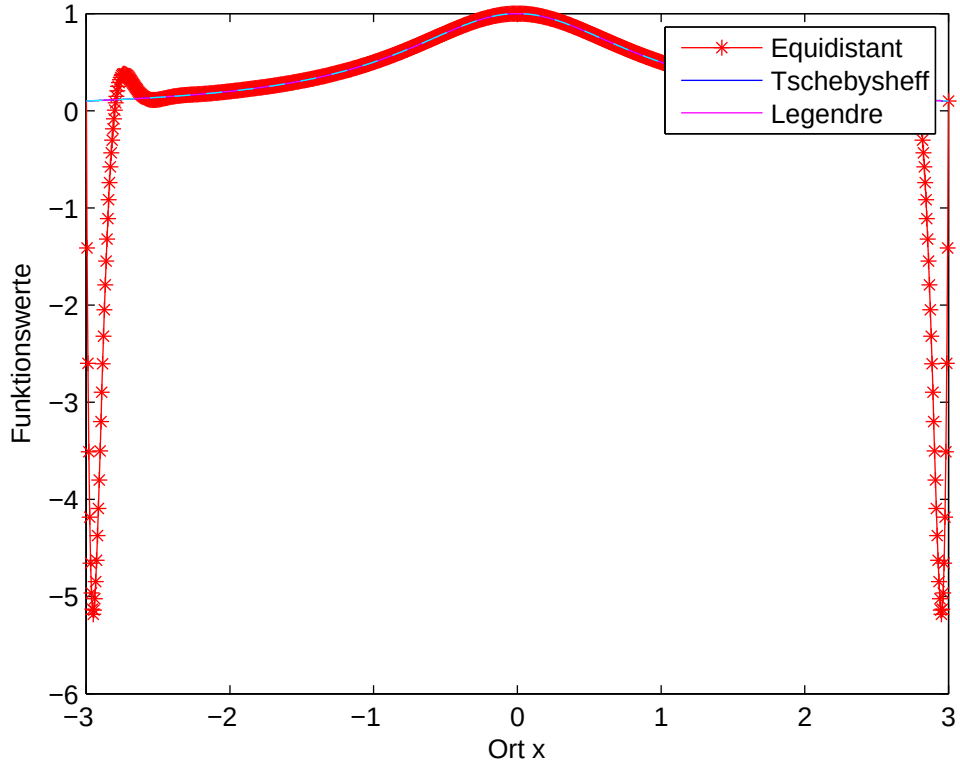

Abbildung 1: Funktionsplots von Interpolierten

Erläuterungen:

1. *Dieses Beispiel zeigt nicht nur, dass eine passende Wahl der Stützstellen die Konvergenz der Interpolationspolynome beschleunigt, sondern auch in gewissen Fällen notwendig ist.*

2. *Dieses Beispiel wird auch "Runges Phänomen"genannt.*

3. *Das Produkt in der Fehlerabschätzung*

$$|f(x) - P[f|x_0, ..., x_n](x)| \leq \frac{||f^{(n+1)}||_\infty}{(n+1)!} \prod_{i=0}^{n} |x - x_i|,$$

wird zwar immer kleiner je mehr äquidistante Stützstellen verwendet werden, insgesamt wird der Fehler aber größer da die Supremumsnorm der Ableitungen immer größer werden. Verwendet man als Stützstellen die Nullstellen der Orthogonal-Polynome, so wird der Term $\prod_{i=0}^{n} |x - x_i|$ *schneller kleiner und der Fehler insgesamt konvergiert gegen 0.*