Dirk Praetorius
Christoph Überhuber

Wintersemester 2006/07
08. Dezember 2006

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 8

Die Aufgaben mit Stern (*) sind bis zur nächsten Übung vorzubereiten und werden dort abgeprüft. Die übrigen Aufgaben dienen nur Ihrer Übung und mir als zusätzliche Grundlage für den Prüfungsstoff in den schriftlichen Tests. Kopieren Sie bitte den Source-Code in ein Unterverzeichnis `serie08` *Ihres Home-Verzeichnisses. Überprüfen Sie bitte vor der Übung, ob Ihre Source-Codes mit* `matlab` *auf der* `cad.zserv.tuwien.ac.at` *interpretiert werden können. In den folgenden Aufgaben sollen im wesentlichen* ***Schleifen*** *geübt werden.*

Aufgabe 66*. Man schreibe eine verbesserte Variante von Bubble-Sort, bei dem man sich die Stelle des letzten Tausches merkt. Ab dieser Stelle müssen die Daten ja bereits sortiert sein. Der nächste Durchlauf braucht also nur noch bis zu dieser Array-Position zu gehen. Zur Beschreibung von Bubble-Sort siehe Aufgabe 26. Speichern Sie den Source-Code unter `bubblesort.m` ins Verzeichnis `serie08`. Sie können Ihren Code mit Hilfe des Matlab-Befehls `sort` verifizieren.

Aufgabe 67*. Der Funktionspointer (sog. *function handle*) einer Funktion `fct` ist in Matlab `@fct`. Er kann mit Hilfe des Befehls `feval` ausgewertet werden: Enthält die Variable `pointer` beispielsweise den Funktionspointer der Funktion `sin`, so sind

```
y = feval(pointer,x)    und   y = sin(x)
```

äquivalent. Man schreibe eine nicht-rekursive Funktion `bisektion`, die das Bisektionsverfahren aus Aufgabe 14 realisiert. Den Programmcode speichere man unter `bisektion.m` ins Verzeichnis `serie08`.

Aufgabe 68*. Zu gegebenen reellen Stützstellen $x_1 < \cdots < x_n$ und Funktionswerten $y_j \in \mathbb{R}$ garantiert die Lineare Algebra ein eindeutiges Polynom $p(t)$ vom Grad $n-1$ mit $p(x_j) = y_j$ für alle $j = 1, \ldots, n$. Nun sei $t \in \mathbb{R}$ fixiert und $p(t)$ gesucht. Man kann $p(t)$ mit dem *Neville-Verfahren* berechnen: Dazu definiere man für $j, m \in \mathbb{N}$ mit $m \geq 2$ und $j + m \leq n + 1$ die Werte

$$p_{j,1} := y_j,$$
$$p_{j,m} := \frac{(t - x_j)p_{j+1,m-1} - (t - x_{j+m-1})p_{j,m-1}}{x_{j+m-1} - x_j}.$$

Es gilt dann $p(t) = p_{1,n}$. Man schreibe eine Funktion `neville`, die den Auswertungspunkt $t \in \mathbb{R}$ sowie die Vektoren $x, y \in \mathbb{R}^n$ übernimmt und $p(t)$ mittels Neville-Verfahren berechnet. Dazu berücksichtige man das folgende schematische Vorgehen

$$\begin{array}{ccccccccccc}
y_1 & = & p_{1,1} & \longrightarrow & p_{1,2} & \longrightarrow & p_{1,3} & \longrightarrow & \cdots & \longrightarrow & p_{1,n} = p(t) \\
 & & & \nearrow & & \nearrow & & & & \nearrow & \\
y_2 & = & p_{2,1} & \longrightarrow & p_{2,2} & & & & \nearrow & & \\
 & & & \nearrow & & & & & & & \\
y_3 & = & p_{3,1} & \longrightarrow & \vdots & & & & & & \\
\vdots & & \vdots & & \vdots & \nearrow & & & & & \\
y_{n-1} & = & p_{n-1,1} & \longrightarrow & p_{n-1,2} & & & & & & \\
 & & & \nearrow & & & & & & & \\
y_n & = & p_{n,1} & & & & & & & &
\end{array} \tag{1}$$

Der mathematische Beweis für diesen Algorithmus folgt in der Vorlesung zur Numerischen Mathematik. Speichern Sie die Funktion unter `neville.m` ins Verzeichnis `serie08`. Sie können den Code testen, indem Sie für ein bekanntes Polynom p als Funktionswerte $y_j = p(x_j)$ wählen.

Aufgabe 69. Man kann das Neville-Verfahren aus Aufgabe 68 so programmieren, dass zur Speicherung der Werte *keine* Matrix $(p_{j,m})_{j,m=1}^{n}$ aufgebaut wird, sondern die gegebenen y_j-Werte geeignet überschrieben werden. Dadurch wird kein weiterer Speicher benötigt.

Aufgabe 70. Eine effiziente Implementierung des einseitigen Differenzenquotienten $\Phi(h)$ aus Aufgabe 53 verwendet die vorherigen Werte $\Phi(h_0), \ldots, \Phi(h_n)$, indem man (theoretisch!) das Interpolationspolynom p_n vom Grad $n-1$ zu den Punkten $(h_j, \Phi(h_j))$ für $j = 1, \ldots, n$ betrachtet, d.h. $p_n(h) \approx \Phi(h)$, und dieses mit dem Neville-Verfahren bei $h = 0$ auswertet. Man bezeichnet dieses Vorgehen als *Richardson-Extrapolation des einseitigen Differenzenquotients*. (Einen Konvergenzbeweis für dieses Verfahren sehen Sie in der Vorlesung zur Numerischen Mathematik.) Mit $h_n := 2^{-n} h_0$ betrachten wir die Folge der $y_n := p_n(0)$. Man schreibe eine Funktion `richardson`, die neben dem Funktionspointer einer Funktion f, den Auswertungspunkt x, die erste Schrittweite $h_0 > 0$ sowie die Toleranz $\varepsilon > 0$ übernimmt und $y_{n+1} \approx f'(x)$ zurückliefert, sobald

$$|y_n - y_{n+1}| \leq \varepsilon \max\{|y_n|, |y_{n+1}|\}$$

gilt.

Aufgabe 71. Für einen stetigen Integranden $f : [a, b] \to \mathbb{R}$ berechnet man das Integral $I := \int_a^b f\,dx$ über geeignete Summen. Bei der *summierten Trapezregel* berechnet man für gegebenes $n \in \mathbb{N}$ und $h := (b-a)/n$ zum Beispiel

$$I_n := \frac{h}{2}\Big(f(a) + 2\sum_{j=1}^{n-1} f(a+jh) + f(b)\Big). \tag{2}$$

Dies ist gerade das Integral über die stetige und stückweise affine Funktion p mit $p(a+jh) = f(a+jh)$. Man schreibe eine Funktion `trapezregel`, die den Wert I_n als Approximation von I berechnet. Man teste die numerische Integration am Beispiel $f(x) = \exp(x)$ auf dem Intervall $[0, 10]$ und gebe abhängig von n den Fehler $|I - I_n|$ tabellarisch aus.

Aufgabe 72. Man implementiere eine Funktion `eratosthenes`, die das *Sieb des Eratosthenes* realisiert. Dies ist ein Algorithmus, mit dem alle Primzahlen bis zu einer bestimmten Zahl `nmax` errechnet werden können (benannt nach dem griechischen Mathematiker Eratosthenes). Der Algorithmus sieht folgendermaßen aus:

- Man streiche aus einer mit der Zahl 2 beginnenden Liste natürlicher Zahlen bis zu einem gewünschten Maximalwert alle Vielfachen der ersten Zahl (also der Zahl 2).
- Wähle, solange es noch höhere Zahlen gibt, die nächsthöhere nicht durchgestrichene Zahl und streiche alle ihre Vielfache.

Der Rückgabevektor `prim` soll dann (mit minimaler Länge) alle Primzahlen $\leq$ `nmax` enthalten. Sie können Ihre Funktion mit Hilfe der Matlab-Funktin `primes` verifizieren.

Aufgabe 73. Schreiben Sie eine Funktion `exponential`, die den Funktionswert $\exp(x)$ approximativ berechnet: Dazu berechnen Sie die Partialsumme

$$S_N(x) := \sum_{j=0}^{N} \frac{x^j}{j!},$$

wobei die Summationsgrenze $N \in \mathbb{N}$ durch das Kriterium

$$\Big|\frac{x^{N+1}}{(N+1)!}\Big| \leq \Big|\frac{x^N}{N!}\Big| \leq \varepsilon$$

für eine gegebene Toleranz $\varepsilon > 0$ bestimmt werde. Intern realisiere man die Berechnung der Summationsglieder $x^j/j!$ möglichst rechenökonomisch. Vergleichen Sie den Fehler $|S_N(x) - \exp(x)|$ für verschiedene Wahlen von $\varepsilon > 0$ und Auswertungspunkten $x \in \mathbb{R}$.