Dirk Praetorius
Christoph Überhuber

Wintersemester 2006/07
Weihnachten 2006

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 9

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 `serie09` *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*** *und* ***Rekursion*** *vertierft werden.*

Aufgabe 74*. Es sei $L \in \mathbb{R}^{n\times n}$ eine untere Dreiecksmatrix mit $\ell_{jj} \neq 0$ für alle $j = 1, \dots, n$. Dann ist L invertierbar, und die Inverse läßt sich wie folgt rekursiv berechnen: Wir schreiben L in Block-Form

$$L = \begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix}$$

mit $L_{11} \in \mathbb{R}^{p\times p}$, $L_{21} \in \mathbb{R}^{q\times p}$ und $L_{22} \in \mathbb{R}^{q\times q}$, wobei $n = p+q$ gilt. Man beachte, dass L_{11} und L_{22} wieder untere Dreiecksmatrizen sind mit $\ell_{jj} \neq 0$. Üblicherweise wählt man $p = n/2$, falls n gerade ist, bzw. $p = (n-1)/2$, falls n ungerade ist. Offensichtlich wird die Inverse L^{-1} dann durch

$$L^{-1} = \begin{pmatrix} L_{11}^{-1} & 0 \\ -L_{22}^{-1} L_{21} L_{11}^{-1} & L_{22}^{-1} \end{pmatrix}$$

gegeben. Man schreibe eine Funktion `invertL`, die die Inverse L^{-1} rekursiv berechnet. Den Source-Code speichere man ins Verzeichnis `serie09`. Sie können die Korrektheit Ihrer Funktion mit Hilfe der Matlab-Funktion `inv` überprüfen.

Aufgabe 75*. Das Integral $\int_a^b f\,dx$ einer stetigen Funktion f wird üblicherweise durch sogenannte Quadraturformeln berechnet. Die *Trapezregel* $I_1(f,a,b)$ sowie die *Simpson-Regel* $I_2(f,a,b)$ sind beispielsweise durch

$$I_1(f,a,b) := \frac{b-a}{2}\big(f(a)+f(b)\big) \quad \text{und} \quad I_2(f,a,b) := \frac{b-a}{6}\Big(f(a)+4\,f\big(\frac{a+b}{2}\big)+f(b)\Big)$$

gegeben. Schreiben Sie eine rekursive Funktion `adquad`, die das Integral $\int_a^b f\,dx$ mittels Simpson-Regel $I_2(f,a,b)$ und iterierter Intervallhalbierung

$$\int_a^b f\,dx = \int_a^{(a+b)/2} f\,dx + \int_{(a+b)/2}^b f\,dx \approx I_2(f,a,(a+b)/2) + I_2(f,(a+b)/2,b)$$

berechnet. Dabei soll der Integrationsbereich jeweils halbiert werden, falls

$$|I_1(f,a,b) - I_2(f,a,b)| > \varepsilon\,\max\{|I_1(f,a,b)|,|I_2(f,a,b)|\} \quad \text{und} \quad |b-a| > h_{\min}$$

ist. Der Funktion sollen neben f, a und b auch die Parameter ε und $h_{\min}$ übergeben werden. Speichern Sie den Source-Code im Verzeichnis `serie09`. Sie können Ihren Code mit Hilfe der Matlab-Funktion `quad` verifizieren.

Aufgabe 76*. Wir betrachten wieder den einseitigen Differenzenquotienten

$$\Phi(h) := \frac{f(x+h)-f(x)}{h} \quad \text{für } h > 0$$

zur Approximation der Ableitung $f'(x) = \Phi(0)$. Für $f \in \mathcal{C}^2$ gilt $|f'(x) - \Phi(h)| = \mathcal{O}(h)$. Ist die Funktion $f : \mathbb{R} \to \mathbb{R}$ nicht glatt genug, so bekommt man lediglich $e_h := |f'(x) - \Phi_h| = \mathcal{O}(h^\alpha)$ für einen Parameter $\alpha \geq 0$, den man *Konvergenzordnung* nennt. Mit dem Ansatz $e_h = ch^\alpha$ und $e_{h/2} = c(h/2)^\alpha$ erhält man durch Auflösen numerische Werte $\alpha := \log(e_h/e_{h/2})/\log(h/(h/2)) = \log(e_h/e_{h/2})/\log(2)$ und $c := e_h/h^\alpha$. Man bezeichnet α in diesem Fall als *experimentelle Konvergenzordnung*. Man erweitere die Funktion `diff` aus Aufgabe 53, sodass in jedem Schritt $n \geq 1$ h_n, $\Phi(h_n)$ und e_{h_n} sowie c und α übersichtlich in tabellarischer Form ausgegeben werden. Dazu berechnen Sie c und α mittels (h_{n-1}, h_n). Für α verwende man Fixpunktdarstellung mit 2 Nachkommastellen (z.B. 1.23), für die übrigen Daten Exponentialdarstellung mit 3 Nachkommastellen (z.B. $1.378e-3$).

Aufgabe 77. Im Allgemeinen ist $f'(x)$ unbekannt, und der Fehler e_h kann deshalb nicht berechnet werden. Mit $\delta_h := |\Phi(h) - \Phi(h/2)|$ und demselben Ansatz wie in Aufgabe 76 zeigt die Dreiecksungleichung $e_h(1-2^{-\alpha}) \leq \delta_h \leq e_h(1+2^{-\alpha})$. Wenn also e_h nicht bekannt ist, kann man die experimentelle Konvergenzordnung über δ_h anstelle von e_h berechnen, d.h. $\alpha := \log(\delta_h/\delta_{h/2})/\log(2)$ und $c := \delta_h/h^\alpha$. Man modifiziere den Source-Code zu Aufgabe 76: Für jeden Schritt $n \geq 2$ berechne man nun α und c basierend auf (h_{n-2}, h_{n-1}, h_n) und gebe h_n, $\Phi(h_n)$, $\delta_{h_{n-1}}$, c und α übersichtlich in tabellarischer Form aus.

Aufgabe 78. Alternativ kann man die Ableitung $f'(x)$ auch durch den zentralen Differenzenquotienten

$$\Phi(h) := \frac{f(x+h) - f(x-h)}{2h} \quad \text{für } h > 0$$

approximieren. Man realisiere dies in einer Funktion `diff2` analog zu Aufgabe 77 und vergleiche die Konvergenzordnungen (und Laufzeiten) von `diff` und `diff2` zur Approximation von $e = \exp(1) = \exp'(1)$.

Aufgabe 79. Wir betrachten das Integral $I := \int_a^b f\,dx$ eines stetigen Integranden $f : [a,b] \to \mathbb{R}$. Für gegebenes $n \in \mathbb{N}$ und $h := (b-a)/n$ sind die *summierte Trapezregel* $I_{1,n}(f,a,b)$ und die *summierte Simpsonregel* $I_{2,n}(f,a,b)$ definiert durch

$$I_{1,n}(f,a,b) := \frac{h}{2} \sum_{j=0}^{n-1} I_1\big(f, a+jh, a+(j+1)h\big) \quad \text{und} \quad I_{2,n}(f,a,b) := \frac{h}{2} \sum_{j=0}^{n-1} I_2\big(f, a+jh, a+(j+1)h\big),$$

vgl. Aufgabe 71 sowie 75. Für gegebenes `nmax` und $n = 1, \ldots,$ `nmax` berechne man die Werte $I_{1,n}(f,a,b)$ bzw. $I_{2,n}(f,a,b)$ und gebe die experimentellen Konvergenzraten aus. — Wie sieht die Formel für die experimentelle Konvergenzrate *in diesem Beispiel* aus?

Aufgabe 80. Man schreibe eine Funktion `merge`, die zwei aufsteigend sortierte Felder a und b so vereinigt, dass das resultierende Feld c ebenfalls aufsteigend sortiert ist, z.B. soll also Aufruf mit $a = (1,3,3,4,7)$ und $b = (1,2,3,8)$ als Ergebnis $c = (1,1,2,3,3,3,4,7,8)$ liefern.

Aufgabe 81. Man schreibe eine rekursive Funktion `mergesort`, die ein Feld a aufsteigend sortiert und das sortierte Feld zurückgibt:

- Hat a Länge ≤ 2, so wird das Feld a explizit sortiert.
- Hat a Länge > 2, halbiere a in zwei Teilfelder a_1 und a_2, rufe `mergesort` für a_1 und a_2 auf und vereinige die sortierten Teilfelder mittels `merge` aus Aufgabe 80.

Was ist der Vorteil von *Mergesort* gegenüber *Bubblesort*?

Aufgabe 82. Man schreibe eine Funktion `primfaktoren`, die zu einer gegebenen Zahl $x \in \mathbb{N}$ ein Feld $y = (y_1, \ldots, y_n)$ zurückliefert, in dem die eindeutigen Primfaktoren von x stehen. Sie können Ihre Funktion mit Hilfe der Matlab-Funktion `factor` verifizieren.

Aufgabe 83. Gegeben seien ein Vektor $x \in \mathbb{R}^n$ und eine Schranke $k \in \mathbb{N}$. Man schreibe eine Funktion `cut`, die aus x alle Glieder x_j mit $|x_j| \geq k$ streicht und als Vektor y zurückliefert, d.h. der Vektor y ist ein gekürzter Vektor x. Mehrere Realisierungen sind denkbar:

- Man geht zunächst x durch und zählt die Anzahl der Folgenglieder $x(j)$ mit $x(j) < k$. Man legt dann y mit geeigneter Länge an und kopiert $x(j)$ nach y, falls $x(j) < k$ gilt.
- Man geht die Folge x von vorne durch, streicht die entsprechenden Folgenglieder und kopiert die nachfolgenden Folgenglieder um eine Stelle nach vorne. Am Ende muss x geeignet gekürzt werden.

Aufgabe 84. Man schreibe eine (rekursive) Funktion `papierschnitt`, die alle Möglichkeiten visualisiert, wie ein Papierbogen der ganzzahligen Länge `laenge` in Papierbahnen der Länge 1 und 2 geschnitten werden kann. — D.h. man stelle eine natürliche Zahl $n =$ `laenge` auf alle möglichen Weisen als Summe $n = \sum_{j=1}^{k} \sigma_j$ mit Summanden $\sigma_j \in \{1,2\}$ dar. Dabei soll die Reihenfolge beachtet werden. Für `laenge=4` gibt es beispielsweise 5 Möglichkeiten:

- $4 = 2+2$
- $4 = 2+1+1$
- $4 = 1+2+1$
- $4 = 1+1+2$
- $4 = 1+1+1+1$

Aufgabe 85. Es sei $U \in \mathbb{K}^{n\times n}$ eine obere Dreiecksmatrix mit $u_{jj} \neq 0$ für alle $j = 1, \ldots, n$. Dann ist U invertierbar, und die Inverse kann rekursiv berechnet werden. Man schreibe eine Funktion analog zu Aufgabe 74.

Aufgabe 86. Nach einer alten Legende gibt es in Hanoi einen Tempel, in dem sich folgende rituelle Anordnung befindet: Es handelt sich um 3 Pfosten und um $n = 64$ goldene Scheiben, die in der Mitte ein Loch haben, so daß sie auf die Pfosten gestapelt werden können. Die 64 Scheiben haben paarweise verschiedenen Durchmesser. Als der Tempel erbaut wurde, wurden diese Scheiben der Größe nach aufsteigend auf den ersten Pfosten gestapelt. Die Aufgabe der Priester in diesem Tempel besteht darin, diese Scheiben systematisch unter Zuhilfenahme des zweiten Pfostens so umzustapeln, daß sich diese am Schluß in derselben Anordnung wie zu Beginn auf dem dritten Pfosten befinden. Dabei sind folgende Regeln zu beachten:

- Die Scheiben dürfen nur einzeln verschoben werden.
- Eine Scheibe darf nie auf einer anderen Scheibe kleineren Durchmessers zu liegen kommen.

Das Ende der Welt, so sagt die Legende, ist durch denjenigen Zeitpunkt vorherbestimmt, an dem die Priester diese Aufgabe erfüllt haben werden.
Ein rekursiver Algorithmus, der dieses Problem löst, läßt sich wie folgt formulieren: Um die obersten m auf Pfosten i befindlichen Scheiben auf Pfosten j zu verschieben, verschiebt man

1. die obersten $m-1$ Scheiben von Pfosten i auf Pfosten $k \notin \{i, j\}$,
2. die größte der besagten m Scheiben von Pfosten i auf Pfosten j,
3. und schließlich die $m-1$ in Schritt 1 auf Pfosten k verschobenen Scheiben auf Pfosten j.

Man beginnt mit $m = n$, $i = 1$ und $j = 3$. Man entwickle eine rekursive Funktion, die diesen Algorithmus implementiert. Jeder einzelne Schritt soll dabei am Display protokolliert werden. Zum Beispiel:

```
Scheibe 1 wandert vom 2. zum 3. Pfosten
```

Zum Testen verwende man $n \ll 64$, z.B. $n = 3, 4, 5$.

Aufgabe 87. Das Neville-Verfahren aus Aufgabe 68 läßt sich ebenfalls rekursiv implementieren. Man schreibe eine rekursive Funktion `rekneville`, die die Auswertung $p_{j,m}(t)$ des Neville-Polynoms $p_{j,m}$ rekursiv berechnet. Ist Rekursion in diesem Beispiel sinnvoll? Was sind die Stärken, was sind die Schwächen?

Aufgabe 88. Zu gegebenen reellen Stützstellen $x_0 < x_1 < \cdots < x_n$ und Funktionswerten $y_j \in \mathbb{R}$ existiert ein eindeutiges Polynom $p(t)$ vom Grad n mit $p(x_j) = y_j$ für alle $j = 0, \ldots, n$. Es ist oft vorteilhaft, das Polynom $p(t)$ in der Newton-Basis darzustellen,

$$p(t) = a_0 + a_1(t - x_0) + a_2(t - x_0)(t - x_1) + \ldots + a_n(t - x_0)\cdots(t - x_{n-1}).$$

Die Koeffizienten $a_0, a_1, \ldots, a_n \in \mathbb{R}$ des Polynoms kann man mit dem *Verfahren der dividierten Differenzen* berechnen. Dazu definiert man für $j, m \in \mathbb{N}_0$ mit $j + m \leq n$ die Werte

$$\begin{aligned} y_{j,0} &:= y_j, \\ y_{j,m} &:= \frac{y_{j+1,m-1} - y_{j,m-1}}{x_{j+m} - x_j}. \end{aligned}$$

Es gilt dann $a_\ell = y_{0,\ell}$. Es soll eine Funktion `newton` geschrieben werden, die bei Übergabe der Vektoren x und y den Koeffizientenvektor $a \in \mathbb{R}^{n+1}$ berechnet. Dazu berücksichtige man das Schema

$$\begin{array}{ccccccccccc}
y_0 & = & y_{0,0} & \longrightarrow & y_{0,1} & \longrightarrow & y_{0,2} & \longrightarrow & \cdots & \longrightarrow & y_{0,n} \\
 & & & \nearrow & & \nearrow & & & & \nearrow & \\
y_1 & = & y_{1,0} & \longrightarrow & y_{1,1} & & & & & & \\
 & & & \nearrow & & & & \nearrow & & & \\
y_2 & = & y_{2,0} & \longrightarrow & \vdots & & & & & & \\
\vdots & & \vdots & & \vdots & \nearrow & & & & & \\
y_{n-1} & = & y_{n-1,0} & \longrightarrow & y_{n-1,1} & & & & & & \\
 & & & \nearrow & & & & & & & \\
y_n & = & y_{n,0} & & & & & & & &
\end{array}$$

Wenn möglich, schreibe man den Pseudo-Code so, dass *keine* Matrix $(y_{j,m})_{j,m=0}^{n}$ aufgebaut wird, sondern die y_j-Werte geeignet überschrieben werden.