Winfried Auzinger
Klaus Felsenstein
Dirk Praetorius

SS 2009
(30. April 2009)

Übungen zur Vorlesung Computermathematik

Serie 7

Die Aufgaben mit Stern (*) sind bis zur Übung in der kommenden Woche vorzubereiten und werden dort abgeprüft. Die übrigen Aufgaben dienen nur Ihrer Übung. Kopieren Sie bitte Ihre Worksheets auf Ihren Account auf `lva.student.tuwien.ac.at`*. Überprüfen Sie vor der Übung, ob Ihre Codes unter Maple auf* `lva` *einwandfrei funktionieren.*

Es wird empfohlen, für jedes Beispiel im Verzeichnis `serie07` *ein eigenes Worksheet mit dem Namen* `aufgabeyy.mws` *anzulegen. Alle zu verfassenden Prozeduren sind zu kommentieren und – je nach Angabe – mit verschiedenen Werten für die Parameter auszutesten.*

Aufgabe 7.1*. Der Maple-Befehl `convert` dient für verschiedenartigste Umwandlungen von Maple-Objekten in eine andere Gestalt. Studieren Sie die Online-Hilfe (`? convert`) und erstellen Sie ein Worksheet, in dem einige wichtige Anwendungen dokumentiert sind.

Beispiel: `? convert/binary`: Konversion einer reellen Zahl in Binärdarstellung.
Probieren Sie aus: `convert(0.1,binary); convert(1/10,binary);` und erläutern Sie.

Aufgabe 7.2*. Schreiben Sie eine Maple-Prozedur

```
Lagrangepoly := proc(ty::listlist,t)
  description "Lagrange-Interpolation";
  ...
```

(vgl. `? type/listlist`, `? Describe`), die zu einer gegebenen Liste `ty` von Wertepaaren `ty[k] = [`t_k, y_k`]` (Länge n, t_k paarweise verschieden angenommen) das zugehörige Lagrange-Interpolationspolynom $p(t)$ vom Grad $\leq n-1$ mit der Eigenschaft

$$p(t_k) \,=\, y_k, \qquad k = 1 \ldots n$$

als Formelausdruck in der Variablen `t` zurückgibt. Man verwende folgende Darstellung für p ('Lagrange-Darstellung'):

$$p(t) \,=\, \sum_{k=1}^{n} y_k \, L_k(t)$$

mit den sogenannten Lagrange-Polynomen $L_k(t)$ vom Grad $n-1$, die durch die Eigenschaft $L_k(t_j) = \delta_{kj}$ eindeutig festgelegt sind. (Man überlege, wie die $L_k(t)$ darzustellen sind.)

Was passiert, wenn eines der t_k in der übergebenen Liste `ty` irrtümlich öfters auftritt?

Man kann die Prozedur auch mit symbolischen Werten füttern und erhält dann das Polynom als Ausdruck in den betreffenden Parametern.

Aufgabe 7.3*. Es seien $p(x) = a_m x^m + \ldots + a_1 x + a_0$ und $q(x) = b_n x^n + \ldots + b_1 x + b_0$ Polynome vom Grad m bzw. n mit ganzzahligen Koeffizienten. Im Folgenden nehmen wir der Einfachheit halber an $m > n$, $a_m = 1$ und $b_n = 1$. Unter *Polynomdivision* versteht man die Bestimmung zweier (eindeutig bestimmter) Polynome $s(x)$ vom Grad exakt gleich $m-n$ und $r(x)$ vom Grad $< n$ mit

$$p(x) \,=\, q(x)s(x) + r(x)\,.$$

Man überlege, wie man die Koeffizienten von s und r mit Hilfe eines Koeffizientenvergleiches systematisch bestimmt; dies zeigt, dass s und r tatsächlich eindeutig bestimmt sind und ebenfalls ganzzahlige Koeffizienten haben.

Darauf basierend schreibe man eine Maple-Prozedur `poldiv(P,Q,x)`, die `P`=$p(x)$ und `Q`=$q(x)$ als Ausdrücke in der Variablen `x` übernimmt und $s(x)$, $r(x)$ wieder als Ausdrücke in der Variablen `x` zurückgibt. (Siehe `?degree`. Auf diverse Typüberprüfungs-Spielereien wird hier verzichtet, d.h. es darf angenommen werden, dass die übergebenen Ausdrücke der obigen Spezifikation entsprechen.)

Beispiel: $p(x) = x^3 + 2x^2 + 4x + 3,\ \ q(x) = x^2 + x + 5 \quad \Rightarrow$

$$s(x) \,=\, x + 1, \quad r(x) \,=\, -2x - 2\,.$$

Also:

```
> P:=x^3+2*x^2+4*x+3: Q:=x^2+x+5:
> S,R:=poldiv(P,Q,x);
```

$$S,R := 1{+}x,\ -2{-}2x$$

Hinweis: Für die Extraktion von Koeffizienten bzw. zum Zweck des Koeffizientenvergleichs verwendet man `coeff(...)` (siehe auch `?collect`). Zur Überprüfung der Ergebnisse siehe `?quo` und `?rem`.

Aufgabe 7.4. Die Maple-Befehle `kernelopts()` und `interface()` dienen der individuellen Konfiguration des Verhaltens zwischen dem Benutzer und dem Kernel bzw. dem Worksheet-Interface. Studieren Sie die Online-Hilfe (`?kernelopts`, `?interface`) und erstellen Sie ein Worksheet, in dem einige wichtige Einstellungen dokumentiert sind.

Aufgabe 7.5*. Ein simples Verfahren zur verlustfreien Datenkompression funktioniert wie folgt. Der Einfachheit betrachten wir nur natürliche Zahlen in Dezimaldarstellung (mit beliebig vielen Ziffern). Wenn eine Ziffer öfters, also n-mal hintereinander auftritt, wird sie nur einmal gespeichert und zusätzlich wird vermerkt, wie oft hintereinander sie auftritt (für $n \geq 2$, nicht für $n = 1$!), z.B.

3111000022 $\quad\rightarrow\quad$ 3, 1 (3 mal), 0 (4 mal), 2 (2 mal).

Eine demgemäß komprimierte Darstellung ist in geeigneter Weise zu codieren. Man überlege die Art der Codierung – es soll natürlich tatsächlich speichereffizienter sein und im Fall 'maximaler Entropie', d.h. wenn keine (oder nur wenige) Wiederholungen auftreten, nicht speicherintensiver sein[1] als die unkomprimierte Darstellung. Im Fall 'minimaler Entropie', z.B. `111111 ... 111111111`, soll die komprimierte Speicherung entsprechend kompakt ausfallen ('entropy coding').

Man denke etwa an eine Liste von Integers, in der die komprimierte Ziffernfolge und die Information über die Wiederholungsraten in geeigneter Weise protokolliert werden. Schreiben Sie zwei entsprechende Prozeduren `encode(...)` und `decode(...)` für Kompression und Dekompression.

Aufgabe 7.6. Eine reelle symmetrische Tridiagonalmatrix

$$A \,=\, \begin{pmatrix} a_1 & b_2 & & & \\ b_2 & a_2 & b_3 & & \\ & \ddots & \ddots & \ddots & \\ & & b_{n-1} & a_{n-1} & b_n \\ & & & b_n & a_n \end{pmatrix}$$

repräsentieren wir in kompakter Weise als `Array` der Dimension `(1..n,1..2)`. Jede Zeile des Arrays besteht aus einer 2 Elementen, die je einem a- bzw. einem b-Wert entsprechen. Ausgehend von dieser Darstellung schreibe man eine Maple-Prozedur `dettridiag(A)`, die eine derartige Struktur als Parameter übernimmt und iterativ die Determinante der entsprechenden Matrix bestimmt. (Die Iteration leitet man sich mit Hilfe des Laplace'schen Entwicklungssatzes her.)

Man verwende diese Prozedur auch dazu, um das charakteristische Polynom $p(\lambda) = \det(A - \lambda\, I)$ explizit als Polynom in der Variablen λ darzustellen.

Aufgabe 7.7. Man schreibe eine rekursive Maple-Prozedur `binary_search(e,l)`, die zu einer gegebenen, als aufsteigend sortiert angenommenen Liste `l` von Maple-Objekten (mehrfaches Auftreten ist erlaubt) eine Position

[1] Den intern benötigten Speicher kann man grob über die Gesamtanzahl der für die Darstellung verwendeten Dezimalziffern bewerten.

angibt, an der das gesuchte Element `e` auftritt. Falls `e` nicht gefunden wird, ist `NULL` zurückzugeben. (Es darf a priori vorausgesetzt werden, dass die Liste korrekt aufsteigend sortiert ist. Bei mehrfachem Auftreten sei es gestattet, irgendeine korrekte Position zurückzugeben.)

Algorithmus: 'Divide and conquer' mittels rekursiver Halbierung der Suchbereiche – verwandt zum Sort/Merge Algorithmus (vgl. VO II).

Die übergebenen Listenelemente müssen sinnvollerweise von verwandtem (d.h. vergleichbarem) Typ sein. Sie müssen aber nicht unbedingt vom Typ `numeric`, sondern können z.B. auch vom Typ `string` sein (Wiener Telefonbuch).

Beispiel:

```
> l:=["Huber","Maier","Mayer","Mayer","Meyer","Müller","Nobody"]:
> binary_search("Mayer",l);
                    3
```

Aufgabe 7.8. Eine *Hash-Tabelle* ist eine Möglichkeit zur Indizierung einer Datenbank aufgrund eines gegebenen Schlüssels (Suchbegriffes). Angenommen, Sie speichern die Daten von maximal N Personen, und zwar die Matrikelnummer m (integer, eindeutig) und der Einfachheit halber nur noch den Familiennamen (String fester Länge, nicht eindeutig) in einem Array der festen Länge N. (Das Array ist unsere 'Datenbank'.) Der Datenbestand soll sich dynamisch ändern können; jedesmal nach dem Schlüssel neu sortieren wäre sehr ineffizient. Andererseits will man Einträge über den Schlüssel schnell wieder auffinden können. Dafür verwendet man z.B. eine *Hash-Funktion*, etwa

$$h(m) := (m \mod N) + 1$$

und speichert den Datensatz zum Schlüsselwert m an der Position $h(m)$.

Implementieren Sie für diese Situation das Eintragen eines Datensatzes und das Auslesen (für gegebenes m) mit Hilfe zweier Maple-Prozeduren. Das Daten-Array ist dabei eine globale Variable mit festem Namen.

Die Zuordnung $m \mapsto h(m)$ ist natürlich normalerweise nicht injektiv (es sei denn, N ist sehr groß – dies würde meist bedeuten: viel größer als für die Datenhaltung erforderlich). Für den Fall einer Kollision (Eintrag bereits 'besetzt') muss daher ein Ersatzindex berechnet werden. Die 'naivste' und einfachste Strategie besteht darin, linear im Datenbestand weiterzusuchen (zyklisch, d.h. wenn man am Ende angelangt ist, fängt man von vorne zu suchen an). und den ersten aufgefundenen freien Platz zu besetzen. Dieser Fall ist natürlich auch beim Auslesen zu berücksichtigen, und es ist eine Fehlermeldung 'Datenbank voll' vorzusehen.

Die Methode funktioniert im Mittel ganz gut, wenn die auftretenden Werte von m halbwegs gleichverteilt sind, N als Primzahl gewählt wird und die Datenbank nicht zu 'vollgestopft' ist.

Siehe dazu auch: `http://de.wikipedia.org/wiki/Hashtabelle`

Aufgabe 7.9. Schreiben Sie eine Prozedur `checkaffine(f,n)`, die für eine gegebene reellwertige Funktion $f(x_1, \ldots, x_n)$ in n Variablen überprüft, ob sie linear (genauer: affin) ist, d.h. ob gilt

$$f(x_1, \ldots, x_n) \equiv a_1 x_1 + \ldots + a_n x_n + b\,.$$

Falls dies zutrifft, sind der Vektor $a = (a_1, \ldots, a_n)$ (z.B. als Liste) und b zurückzuliefern. Andernfalls wird `FAIL` zurückgegeben (letzteres kann beim Aufruf z.B. mittels `evalb(checkaffine(f,n)=FAIL)` abgefragt werden).

Aufgabe 7.10. Implementieren Sie die einfachste Variante eines iterativen Verfahren zum Aufsuchen einer Minimalstelle einer nichtlinearen reellwertigen Funktion $f(x, y)$ (*Gradientenverfahren*):

Für einen gegebenen Startwert (x_0, y_0) geht man iterativ wie folgt vor: Man bestimmt den *Gradienten* $\nabla f(x_i, y_i) = (\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y})(x_i, y_i)$ (das ist die Richtung des lokal steilsten Anstiegs von f) und setzt

$$(x_{i+1}, y_{i+1}) := (x_i, y_i) - \alpha_i \nabla f(x_i, y_i). \tag{1}$$

Dies ist ein simpler und naheliegender Ansatz ($-\nabla f$ ist die lokal steilste Abstiegsrichtung) – das Problem besteht in der Wahl des Parameters α_i (Schrittlänge). Eine eher noch naive Variante lautet wie folgt: Probieren Sie $\alpha_i := 1, \frac{1}{2}, \frac{1}{4}, \ldots$ und testen Sie auf Abstieg, d.h.

$$?\ f((x_i, y_i) - \alpha_i \nabla f(x_i, y_i)) < f(x_i, y_i)\ ? \tag{2}$$

Sobald das erfüllt ist, akzeptiert man den neuen Wert gemäß (1) und macht iterativ weiter.

- Erfolgreiche Beendigung sobald $\|\nabla f(x_i, y_i)\|_2 \leq$ `tol`, (warum ?); (x_i, y_i) wird zurückgegeben; es sei denn:
- Fehler-Exit (`error ...`) falls
  - eine vorgegebene Maximalanzahl an Iterationsschritten überschritten wird (Parameter `maxiter`);
  - die Abstiegsbedingung (2) nicht erfüllbar ist, weil eines der α_i einen vorgegebenen Minimalwert `minsteplength` unterschreitet.

Dies ist in einer Prozedur `graditer(f,x[0],y[0],maxiter,minsteplength)` zu implementieren. Überlegen Sie sich selbst ein paar Beispiele.

Diese Aufgabe ist beliebig ausbaufähig (z.B. optionale Protokollierung des Konvergenzverlaufes).