Winfried Auzinger
Klaus Felsenstein
Dirk Praetorius

SS 2011
(10. Mai 2011)

Übungsaufgaben zur VL Computermathematik

Serie 9

Aufgabe 9.1*. Gegeben sei eine Ellipse (Darstellung in Polarkoordinaten:)

$$P(\varphi) = \begin{pmatrix} x(\varphi) \\ y(\varphi) \end{pmatrix} = \begin{pmatrix} a\cos\varphi \\ b\sin\varphi \end{pmatrix}, \quad 0 \le \varphi < 2\,\pi.$$

Erstellen Sie eine Prozedur `kd(a,b,phi)`, die zu einem gegebenen Punkt $P(\varphi)$ auf der Ellipse die beiden zugehörigen konjugierten Durchmesser ermittelt: Der eine ist der durch das durch den Nullpunkt verlaufende Geradenstück, das zum Tangentialvektor $P'(\varphi)$ parallel verläuft, und der andere ist das Geradenstück, das durch 0 und $P(\varphi)$ verläuft. Verwenden Sie diese Prozedur dazu, um die Ellipse und die beiden konjugierten Durchmesser gemeinsam in einem Plot darzustellen. (`?plot/parametric; ?plots[display]`)

Aufgabe 9.2*. Eine $n \times n$-'Kreuz-Matrix' X (n ungerade) der Gestalt (Beispiel: $n = 7$, allgemein analog)

$$A = \begin{pmatrix} x_1 & & & & & & x_7 \\ & x_2 & & & & x_6 & \\ & & x_3 & & x_5 & & \\ & & & x_4 & & & \\ & & x_5 & & x_3 & & \\ & x_6 & & & & x_2 & \\ x_7 & & & & & & x_1 \end{pmatrix}$$

ist durch n gegebene Zahlen $x_1, \ldots, x_n$ definiert. Mit Hilfe des Laplace'schen Entwicklungssatzes leite man eine rekursive Prozedur

`kreuzdet(x)`

her, die die Determinante einer derartigen Matrix berechnet. Die Matrix wird zu diesem Zweck nicht explizit abgespeichert, sondern das eindimensionale Array `x` spezifiziert die Matrix über die Werte $x_1, x_2, \ldots, x_n$.

Stellen Sie mit Hilfe dieser Prozedur eine Vermutung auf, wie die allgemeine Formel für die Determinante lautet. (Möglichst einfache Darstellung erwünscht. Der Beweis dieser Formel ist eine Übung in vollständiger Induktion, ist jedoch nicht Teil dieser Aufgabe.)

Ein zweiter Blick auf die Problemstellung zeigt, dass es eigentlich viel einfacher geht. Erkennen Sie, wie? (Hinweis: Ähnlichkeitstransformation.) So gesehen ist dies ein Beispiel dafür, dass es manchmal besser sein kann erst nachzudenken, bevor man losprogrammiert. Die Programmieraufgabe ist trotzdem sinnvoll (nette Rekursion).

Aufgabe 9.3*. Erstellen Sie eine rekursive Maple-Prozedur `bsearch(e,l,opt)`, die zu einer gegebenen, als strikt monoton aufsteigend angenommenen Liste `l` von (d.h. `l` erfüllt die Eigenschaft `l[1] < l[2] < ...`) die Position `p` angibt, an der das gesuchte Element `e` auftritt. (Falls `e` nicht gefunden wird, ist mit `error "`*Fehlermeldung*`"` abzubrechen.

Falls der Parameter `opt` den Wert 0 hat oder beim Aufruf weggelassen wird, wird nur nur die Position von `p` von `e` zurückgegeben. (Die Anzahl der aktuell übergebenen Parameter ermittelt man mittels `nargs`.) Andernfalls wird die Liste `[p,e]` zurückgegeben.

Vorgangsweise: 'Divide and conquer' mittels rekursiver Halbierung des Suchbereiches – verwandt zum Sort/Merge Algorithmus (vgl. Teil III), aber einfacher.

Anmerkung: Die übergebenen Listenelemente müssen sinnvollerweise von verwandtem (d.h. vergleichbarem) Typ sein, so dass eine natürliche Ordungsrelation existiert. Sie müssen aber nicht unbedingt vom Typ `numeric` (z.B. `integer`) sein, sondern können z.B. auch vom Typ `string` sein (Strings gelten als lexikografisch angeordnet, wie im Telefonbuch).

Beispiel:

```
> l := ["aber","Müller","du","bist","Kuh"]:
> bsearch("du",l);
                                3
> bsearch("Kuh",l,1);
                                [5,"Kuh"]
```

Aufgabe 9.4*. Definieren Sie eine Funktion `B := (t,i,n) -> ...`, die die sogenannten *Bernstein-Polynome* repräsentiert:

$$B_{i,n}(t) := \binom{n}{i} t^i (1-t)^{n-i} \quad (0 \le i \le n).$$

Angenommen, Sie wollen eine konkrete Funktion $B_{i,n}(t)$ (mit konkreten Werten für i und n) an eine Prozedur übergeben, die als Argument eine Funktion $p(t)$ einer einzigen Variablen t erwartet. Das Problem dabei: Eigentlich ist `B(t,i,n)` eine Funktionen von drei Variablen. Was also tun?

– Dafür kann man z.B. `unapply` einsetzen.[1] Verwenden Sie diesen Befehl, um für konkret gegebene Werte von i und n eine Funktion $p(t)$ zu basteln, die zu $B_{i,n}(t)$ äquivalent (aber eben nur eine Funktion der einzigen Variablen t) ist.

– Alternative (eleganter): `?[r]curry`. Lesen Sie nach und realisieren Sie auch diese Variante.

In dieser Weise können Sie ein konkretes Bernstein-Polynom an ihre (zu erstellende) Prozedur `mincoeff(p)` übergeben, die für eine gegebene polynomiale Funktion p das kleinste k ermittelt, für das t^k in $p(t)$ mit einem Koeffizienten $\neq 0$ auftritt ($k \ge 0$). Achtung auf korrekte Berücksichtigung des Trivialfalles $p(t) = 0$ (Rückgabewert z.B. `NULL`).

Hinweis: `?collect; ?coeff`

Aufgabe 9.5. Fortsetzung von Aufgabe 2: Erstellen Sie mittels Modifikation von `kreuzdet` eine rekursive Prozedur

`kreuzcp(x)`

die das charakteristische Polynom $p\colon \lambda \mapsto \det(\lambda I - X)$ als *Funktion* zurückgibt. Wie lautet eine möglichst einfache Darstellung?

Anmerkung: Der Rückgabewert ist eine Funktion. Man kann dieser z.B. dann einen eigenen Namen geben und sie normal verwenden: `p:=kreuzcp(x);` Auswertung an einer Stelle `lambda` mit `p(lambda)`. Der Ableitungsoperator `D` kann mit derart definierten Funktionen jedoch nicht immer umgehen, z.B. wenn in der definierenden Prozedur ein `simplify` vorkommt. (Ausprobieren.)

Testen Sie auch, wie Maple reagiert, wenn Sie für ein vorgegebenes Array `x` die Funktion `p` mittels `p:=kreuzcp(x)` definieren und danach `x` verändern. Ändert sich dann auch die Funktion `p`? (Das Verhalten bei solchen 'Rückwärts-Einsetzungen ist nicht immer a priori klar; besser man probiert es aus.)

[1] Generell dient `unapply` dazu, einen beliebigen Ausdruck in irgendwelchen Variablen (typischerweise das Ergebnis einer zuvor durchgeführten Berechnung) in eine Funktion in einer oder mehrerer dieser Variablen umzuwandeln.

Aufgabe 9.6. Die Auzinger'sche Funktionenfolge $\omega_n(x)$ ist rekursiv definiert als

$$\omega_0(x) := \frac{e^x - 1}{x}\,; \quad \omega_n(x) := \frac{1}{x}\Big(-1 + \sum_{k=0}^{n-1} \frac{\omega_k(x)}{n-k}\Big), \quad n \geq 1.$$

Erstellen Sie eine rekursive Prozedur `omega(x,n)`, die für konkretes n den entsprechenden Formelausdruck für[2] $\omega_n(x)$ zurückliefert. (Ein typischer Fall für `option remember`. Für eine einfache Darstellung verwenden Sie intern `simplify`.)

Checken Sie mehrere Werte $n = 0, 1, 2, 3, \ldots$ und verifizieren Sie mit Hilfe von Maple, dass diese Funktionen an der Stelle $x = 0$ stetig fortsetzbar sind (`?limit`). Dann versuchen Sie diese Funktionen über dem Intervall $[0, 1]$ zu plotten. Ab einem gewissen Wert von n erhalten Sie nichts vernünftiges, obwohl diese Funktionen ganz 'harmlos' (unendlich oft differenzierbar, glatt) sind. Der Effekt hängt von der eingestellten Rechengenaugkeit ab (`? Digits`). Erhöhung des Wertes für `Digits` hilft – ausprobieren!

Hinweis: Das Problem besteht darin, dass die Auswertung der $\omega_n(x)$ für sehr kleine x auf einen Ausdruck des Typs $a - b$ führt, wobei a, b extrem groß, aber $a - b$ klein ('Auslöschungseffekt). Für eine verlässliche Auswertung eines derartigen Ausdruckes benötigt man eine höhere Rechengenauigkeit.

Anmerkung: Einen expliziten Formelausdruck für $\omega_n(x)$ (für allgemeine $n \in \mathbb{N}$) anzugeben ist sehr trickreich und wird hier nicht verlangt (siehe auch Formel (*)); das geht mit Maple auch nicht direkt. Die Aufgabe stammt aus der numerischen Praxis: de facto dient die obige Rekursion für die einfache rekursive Auswertung der (nicht ganz einfach gebauten) Integrale

$$\omega_n(x) = (-1)^n \int_0^1 e^{(1-\theta)x} \binom{-\theta}{k}\, d\theta, \qquad (*)$$

die bei gewissen numerischen Lösungsverfahren für Differentalgleichungsprobleme benötigt werden. Wenn Sie wollen, können Sie überprüfen, dass diese $\omega_n(x)$ für $n = 0, 1, 2, 3, \ldots$ tatsächlich die obige Rekursion erfüllen.

Aufgabe 9.7. Fortsetzung von Aufgabe 6: Praktikable Approximation der $\omega_n(x)$ in der Nähe von $x = 0$.

Entwickeln Sie für $n = 0, 1, 2, 3, \ldots$ die Funktionen $\omega_n(x)$ in eine Taylorreihe um die Stelle $x = 0$. Achtung: `taylor` kann damit nicht umgehen, weil formal negative Potenzen von x auftreten (die sich allerdings bei der Entwicklung wegheben); verwenden Sie `series` statt `taylor`. In dieser Weise erhalten Sie ein Taylorpolynom mit Restglied (den gewünschten Grad kann man bei `series` vorgeben). Dann verwenden Sie `convert(...,polynom)`, um das Restglied zu entfernen.

Die so entstehenden Taylor-Approximationen (Taylor-Polynome) eignen sich jetzt dazu, um auch bei niedriger Rechengenauigkeit (z.B. `Digits=16`, entspricht etwa der Standard - double Genauigkeit wie in MATLAB) die Funktionen in der Nähe von $x = 0$ grafisch darzustellen. Weiter weg von 0 funktioniert der normale Plot von $\omega_n(x)$, und man kann `plots[display]` verwenden, um den kompletten Plot auf $[0, 1]$ zu generieren.

Das Problem besteht dabei in der Wahl des Grades der Taylorpolynome in Abhängigkeit von n und in der Wahl des Umschaltpunktes (Wechsel von einem Plot zum anderen). Versuchen Sie für $n = 0, 1, 2, 3, \ldots$ mit einem möglichst niedrigen Polynomgrad auszukommen und probieren Sie Umschaltpunkte aus. Für größeres n muss man den Umschaltpunkt weiter weg von 0 wählen, was jedoch auch bedeuten kann, dass man im linken Teil einen höheren Polynomgrad wählen muss. Dies ist nicht leicht zu automatisieren; gehen Sie experimentell vor (trial and error), um Plots zu generieren, die die gleiche optische Qualität aufweisen wie die aus Aufgabe 6.

---

[2] Oder: `omega(n)` mit Funktion als Rückgabewert, vgl Aufgabe 5. Auswertung an einer Stelle `x` dann z.B. mit `omega(3)(x)`.

Aufgabe 9.8. Die Funktionen Folge $\varphi_n(x)$ sei ebenfalls rekursiv definiert (ähnlich wie in Aufgabe 6, aber einfacher):

$$\varphi_0(x) := e^x\,; \quad \varphi_n(x) = \frac{1}{x}\left(\varphi_{n-1}(x) - \frac{1}{(n-1)!}\right), \quad n \geq 1.$$

Erstellen Sie eine rekursive Prozedur `phi(x,n)`, die für konkretes n den entsprechenden Formelausdruck [3] für $\varphi_n(x)$ zurückliefert. Diese Funktionen sind ebenfalls unendlich oft differenzierbar (auch an der Stelle $x = 0$), und beim Plotten besteht ein ähnliches Problem wie bei obigen $\omega_n(x)$.

Die Aufgabenstellung ist hier aber nicht der plot (das geht genauso wie in Aufgabe 6 bzw. 7), sondern die Identifikation dieser Funktionen. Sehen Sie sich diese Funktionen für $n = 0, 1, 2, 3, \ldots$ an und stellen Sie eine Vermutung darüber an, wie sie interpretiert werden können. Falls Sie es nicht erkennen – macht nichts. Verifizieren Sie für mehrere Werte von n, dass die Funktionen

$$\varphi_n(x) = \frac{1}{(n-1)!}\int_0^1 e^{(1-\theta)x}\,\theta^{n-1}\,d\theta, \quad n = 0, 1, 2, \ldots$$

genau die obige Rekursion erfüllen, d.h. das sind unsere gesuchten Funktionen. Können Sie sie jetzt interpretieren? (Analysis 1.)

Testen Sie auch, ob für Maple für allgemeine $n \in \mathbb{N}$ (`assume(n,posint)`) die Integraldarstellungen für $\varphi_n(x)$ auswerten kann und ob `simplify` es schafft, für allgemeines n die Gültigkeit der Rekursionsformel zu verifizieren (quasi als 'Computerbeweis'). Maple 14 und die neue Version Maple 15 können das; vermutlich auch Maple 12 (derzeitige Version am lva-Server, testen Sie selbst). Abschließend könnten Sie versuchen, das selbst zu beweisen (mittels partieller Integration). Das ist nicht schwierig, aber wie bei den meisten Induktionsbeweisen muss man die Lösung bereits kennen (oder richtig vermuten).

[3] Oder: `phi(n)` mit Funktion als Rückgabewert, vgl Aufgabe 5 und 6. Auswertung an einer Stelle `x` dann z.B. mit `phi(4)(x)`.