Winfried Auzinger
Dirk Praetorius

SS 2012
30. April 2012

Übungsaufgaben zur VU Computermathematik

Serie 7

Aufgabe 7.1*. Erstellen Sie eine rekursive Prozedur `bsearch(n,L)`, die die Position einer gegebenen Zahl `n` $\in \mathbb{N}$ in einer Liste `L` ermittelt. Es wird davon ausgegangen, dass `L` eine Liste von unterschiedlichen, ansteigend sortierten Zahlen ist.

Algorithmus: rekursiv. `bsearch` teilt die Liste in zwei (etwa) gleich lange Hälften und sucht rekursiv entweder links oder rechts weiter. Sobald `n` gefunden wird, wird seine Position in `L` als Wert zurückgegeben. Ansonsten wird 0 zurückgegeben.

Anmerkung: Der Fall `L=[]` (leere Liste) soll korrekt berücksichtigt werden.

Aufgabe 7.2*. Programmieren Sie die *Newton-Iteration*

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}\,, \quad i = 0, 1, 2, \ldots$$

zur näherungsweise Lösung einer nichtlinearen Gleichung $f(x) = 0$ (in Gleitpunktarithmetik, `evalf` verwenden).

Die entsprechende Prozedur `niter(f,x0,tol,maxit)` führt ausgehend von der Startnäherung `x0` maximal `maxit` Newton-Schritte durch. Sobald für das Residuum $f(x_i)$ gilt $|f(x_i)| \leq$ `tol`, wird die Iteration beendet. In diesem Fall wird

```
[x,res,it]
```

zurückgegeben, wobei `x` $= x_i$, `res` $= f(x_i)$ und `it` $= i$.

Bei Nicht-Konvergenz (maximale Anzahl der Iterationsschritte erreicht, aber Konvergenzbedingung nicht erfüllt), geben Sie (als 'Warnung') für `it` z.B. den Wert `infinity` zurück.

2 Beispiele: Berechnen Sie $x = \sqrt{a}$ $(f(x) = x^2 - a = 0)$ und $x = 1/a$ mit Hilfe der Newton-Iteration. Für den Fall $1/a$ wird gefordert, dass bei Ausführung der Newton-Iteration nur Addition/Subtraktion und Multiplikation verwendet werden dürfen. (Überlegen Sie, wie Sie das angehen: Welche Gleichung $f(x) = 0$ lösen Sie?)

Aufgabe 7.3*. Die Maple-Funktion `fsolve` ist ein generelles Tool zur numerischen Lösung von Gleichungen bzw. Gleichungssystemen (sie verwendet intern je nach Typ der Gleichung verschiedene Verfahren, z.B. das Newton-Verfahren). Verwenden Sie `fsolve`, um folgenden Kurvenverfolgungsalgorithmus zu implementieren:

Gegeben sei eine reellwertige Funktion $f(x, y)$. Dann definiert die Gleichung $f(x, y) = 0$ in impliziter Weise eine Kurve in der Ebene. Wir nehmen der Einfachheit halber an, dass überall gilt $\frac{\partial f}{\partial y} \neq 0$. Kennt man einen Punkt (x_0, y_0) auf der Kurve, dann besagt der *Hauptsatz über implizite Funktionen*, dass die Gleichung $f(x, y) = 0$ in einer Umgebung von (x_0, y_0) nach y auflösbar ist, d.h., es existiert eine Funktion $y = g(x)$, deren Graph genau die gegebene Kurve ist. Also gilt $f(x, g(x)) \equiv 0$, und mit Hilfe der Kettenregel gilt dann an jedem Kurvenpunkt:

$$0 = \frac{d}{dx} f(x, g(x)) = \frac{\partial f}{\partial x} + \frac{\partial f}{\partial y}\, g'(x)\,,$$

und somit

$$g'(x) = -\frac{\frac{\partial f}{\partial x}}{\frac{\partial f}{\partial y}}\,. \tag{1}$$

Sei nun (x_0, y_0) ein Startpunkt auf der Kurve. Man wählt eine Schrittweite Δx und geht, unter Verwendung von (1) an der Stelle (x_0, y_0) entlang der Tangente um Δx weiter. Dies führt auf einen neuen Punkt $(x_1, \tilde{y}_1)$ auf der Tangente, mit $x_1 = x_0 + \Delta x$. Dieser Punkt liegt nicht auf der Kurve, aber in der Nähe. Lösen Sie nun die Gleichung $f(x_1, y) = 0$ mit Hilfe von `fsolve`, wobei sie für y die Startnäherung $\tilde{y}_1$ angeben (Hilfe dazu: `? fsolve/details`). Die (numerische) Lösung $y =: y_1$ ergibt dann einen neuen Kurvenpunkt (x_1, y_1). Dann macht man von diesem Punkt ausgehend iterativ weiter.

Beispiel: Wenden Sie Ihren Algorithmus auf die durch $(x+y)^3 = (x-y)$ definierte Kurve an. Starten Sie bei $(x_0, y_0) = (-1, 0)$ und verfolgen Sie die Kurve über das Intervall $x \in [-1, 1]$ hinweg. Wählen Sie $\Delta x = 0.1$ (ggf. auch kleiner). Speichern Sie alle (x_i, y_i) in einer Liste und generieren Sie dann eine entsprechende plot-Struktur:[1] `p1:=pointplot(...)`.

Anmerkung: Ein allgemeinerer und adaptiver Algorithmus ist in `implicitplot` implementiert (siehe Aufgabe 7.10). Generieren Sie die plot-Struktur für die Kurve mittels `p2:=implicitplot(...)`, und stellen Sie `p1` und `p2` mittels `display` in einer gemeinsamen Grafik dar.

Aufgabe 7.4*. Für Argumente einer Prozedur kann man zulässige Typen festlegen. Beispiel:

```
proc(n::posint)
```

akzeptiert bei einem Aufruf nur einen Wert `n` $\in \mathbb{N}$ (ansonsten wird mit Fehlermeldung abgebrochen). Auch Mengen von Typen können angegeben werden. Beispiel:

```
proc(x::{numeric,string})
```

(`x` muss eine irgendeine konkrete Zahl oder eine Zeichenkette sein).

Verschiedene zulässige Typen (unabhängig davon, ob man das `::` Konstrukt verwendet oder nicht) bedeuten in vielen Fällen, dass der Algorithmus je nach übergebenem Typ unterschiedlich abläuft (z.B. symbolisch oder numerisch). Dazu fragt man mit Hilfe der Boole'schen Funktion `is` den Typ ab:

`is(`*variable,type*`)`

liefert `true` oder `false`.

Verwenden Sie dies für eine Prozedur `plus(x,y)`. Berücksichtigen Sie folgende zulässigen Fälle:

- `x,y` vom Typ {`name,numeric`}: Gebe `x+y` zurück.
- `x,y` vom Typ `string`: Gebe den zusammengesetzten String zurück (recherchieren Sie selbst, wie der betreffende Befehl lautet).
- `x,y` vom Typ `set`: Gebe `x` $\cup$ `y` zurück.
- `x,y` vom Typ `list`: Gebe `[[x[1],y[1]],[x[2],y[2]],...]` zurück.

In allen anderen Fällen (auch im Fall Listen unterschiedlicher Länge oder einer leeren Liste) wird `NULL` zurückgegeben.

Aufgabe 7.5*. Man kann zwar Dezimalzahlen mittels `convert` in Binärformat umwandeln,[2] aber Arithmetik in binärer Darstellung ist in Maple nicht direkt unterstützt. Erstellen Sie eine Prozedur `badd(m,n)`, die zwei gegebene Zahlen (mit Ziffern nur 0 oder 1) als Binärzahlen interpretiert, binär addiert, und in gleicher Weise zurückgibt, dh. als 'Dezimalzahl' mit 0 und 1 als Ziffern, die der Binärdarstellung des Ergebnisses entspricht.

Dazu ist es nützlich, zunächst zwei Hilfsprozeduren zu schreiben: Eine für die Extraktion der k-ten Stelle einer gegebenen Zahl (verwende `iquo, mod`), und eine für einstellige Binäraddition, mit Übertrag (für den Fall `1+1=10`).

---

[1] Aktivieren Sie das package `plots` mittels `with(plots);`

[2] Beispiel: `convert(9,binary)` ergibt `1001`. Umgekehrt: `convert(1001,decimal,binary)` ergibt `9`.

Zum Testen verwenden schreiben Sie eine Prozedur, die mittels Sie `convert` dasselbe macht. (Man müsste die Binäraddition also eigentlich nicht programmieren, aber das gilt ja für andere Übungsaufgaben auch.)

Beispiel:

`badd(1001,1100011)` ergibt `1101100` $(9 + 99 = 108)$.

Achtung: `iquo, mod` arbeiten mit Dezimalstellen, aber es funktioniert trotzdem, wenn Sie es richtig machen.

Aufgabe 7.6. Eine symmetrische Tridiagonalmatrix

$$T_n = \begin{pmatrix} a_1 & b_1 & & & & \\ b_1 & a_2 & b_2 & & & \\ & b_2 & a_3 & b_3 & & \\ & & \ddots & \ddots & \ddots & \\ & & & b_{n-2} & a_{n-1} & b_{n-1} \\ & & & & b_{n-1} & a_n \end{pmatrix}$$

stellen wir intern mittels einer Liste `T` bestehend aus zwei Listen `a` und `b` dar. Zu berechnen sei $d_n := \det(T_n)$. Leiten Sie zunächst mit Hilfe des Laplace'schen Entwicklungssatzes eine Rekursionsformel der Gestalt

$$d_n = \xi_{n-1}\, d_{n-1} + \eta_{n-2}\, d_{n-2}$$

her, wobei $d_{n-k} = \det(T_{n-k})$ (mit den entsprechenden Untermatrizen), und ξ_{n-1}, η_{n-2} hängen von den Matrixeinträgen ab. Implementieren Sie Ihre Formel als Schleife in einer Prozedur `det3diagsymm(T)`. Verwenden Sie diese auch, um eine Prozedur `charpol3diagsymm(T,x)` zu erstellen, die das charakteristische Polynom $p(x) := \det(T_n - x\, I_n)$ auswertet (x ein Variablenname oder eine konkrete Zahl).

Falls die Längen der Listen übergebenen Listen `a,b` inkompatibel sind, brechen Sie mit einer Fehlermeldung ab. Für diesen Zweck gibt es eine Alternative zu `return`:

`error("`*Fehlermeldung*`");`

Zum Testen können Sie das package `LinearAlgebra` verwenden (wird in VO später genauer besprochen; Aktivierung mit `with(LinearAlgebra);`). Stellen Sie ihre Testmatrix T_n als `Matrix`[3] dar und werten Sie mittels `Determinant`, `CharacteristicPolynomial` aus. Beispiel:

```
> T := Matrix([[a,1,0],
               [1,a,1],
               [0,1,a]]):
> Determinant(T);
               a^3 - 2*a
> CharacteristicPolynomial(T,x);
               x^3 - 3*a*x^2 - (2-3*a^2)*x - a^3+2*a
```

Aufgabe 7.7. Fortsetzung von Aufgabe 7.6:

Verwenden Sie `unapply`, um eine Prozedur `fcharpol3diagsymm(T,x::name)` zu erstellen, die das charakteristische Polynom nicht als Formelausdruck in `x`, sondern als *Funktion* zurückgibt. Wenn Sie es korrekt durchführen, erhalten Sie für obiges Beispiel z.B.:

```
> p := fcharpol3diagsymm(T,x):
> p(y);
          y^3 - 3*a*y^2 - (2-3*a^2)*y - a^3+2*a
```

[3] Eine spezielle zweidimensionale Datenstruktur, speziell für Matrizen aus der Linearen Algebra. (Es gibt auch `Vector`.)

Aufgabe 7.8. Erstellen Sie eine Prozedur `multint(f,X,H)`, die zu einer gegebenen multivariaten Funktion, einer gegebenen Liste `X` (Länge n) von Variablennamen und einer gleich langen Liste `H` von Intervallen I_k das n-fache bestimmte Integral über $I_1 \times I_2 \times \ldots \times I_n$ zurückgibt. Integriert wird entlang der Listen.

Beispiel:

```
> f := (a,b,c,d)->a*b^2*c^3*d^4:
> X := (a,b,c,d):
> H := [[0,1],[0,1],[0,1],[0,1]]:
> multint(f,X,H); # Integral über 4-dimensionalen Hypercube
                  1/120
> int(int(int(int(f(a,b,c,d),a=0..1),b=0..1),c=0..1),d=0..1); # zum Vergleich
                  1/120
```

Aufgabe 7.9. Hat man eine Funktion oder Prozedur mit mehreren Argumenten definiert, z.B. $f(x, y)$, so kann es vorkommen, dass man diese für weitere Verwendung in eine Funktion mit weniger Argumenten umwandeln will, z.B. $g(x) := f(x, y_0)$ mit einem fest vorgegebenen Wert $y = y_0$. Studieren Sie die Dokumentation von `curry` und finden Sie heraus, wie man das macht. Siehe auch `rcurry`.

Probieren Sie es zunächst aus: Definieren Sie irgendeine Funktion $f(x, y, z)$ mit zwei Argumenten als Prozedur und basteln Sie zwei weitere Funktionen $g(x) := f(x, a, b)$ und $h(x, y) = f(x, y, b)$ für vorgegebene feste Werte a bzw. b (diese müssen nicht unbedingt numerisch angegeben werden).

Anwendung: Erstellen Sie eine Funktion oder Prozedur `p(a,b,x,y)`, die ein bivariates Polynom der Gestalt

$$p(x, y) = \sum_{i=1}^{m} \sum_{j=1}^{n} a_i \, b_j \, x^i \, y^j$$

definiert. Dabei werden für `a` und `b` Listen erwartet, und es ist $m =$ `nops(a)`, $n =$ `nops(b)`.

Geben Sie dann zwei konkrete (z.B. numerisch gegebene) Listen `a` und `b` vor und basteln Sie eine Funktion `q(x,y)` für das betreffende konkrete bivariate Polynom, und zwei weitere Funktionen `q[1](x)` und `q[2](y)`, bei denen ein zusätzlich ein fester Wert für `y` bzw. `x` vorgegeben wird (rekursive Anwendung von `[r]curry`.) Testen Sie.

Zusatzfrage:
Kann der Ableitungsoperator `D` mit einer mittels `[r]curry` definierten Funktion umgehen? Testen Sie.

Aufgabe 7.10. Sehen Sie sich das `plots`-package an (Aktivierung mit `with[plots];`), probieren Sie einige der darin enthaltenen Funktionen aus, und führen Sie das in der Übung vor.

Insbesondere interessant:

- `implicitplot, implicitplot3d` (für implizit definierte Kurven bzw. Flächen; 3D kann unter Umständen sehr rechenaufwendig sein),
- `complexplot` (für Kurven in der komplexen Ebene),
- `animate, animate3d, animatecurve` (für kleine Videos). Es erscheint jeweils ein Startbild, und die Anzeige der Animation steuern Sie über ein Kontextmenü (rechte Maustaste).

... Und vieles mehr.