Winfried Auzinger
Dirk Praetorius

SS 2012
8. Mai 2012

Übungsaufgaben zur VU Computermathematik

Serie 8

Vorbemerkung: Für Matrix-Matrix-Multiplikation und Matrix-Vektor-Multiplikation (Datentyp `Matrix`, `Vector`) verwendet man den nichtkommutativen Operator $\cdot$ (Punkt), nicht $*$. $+$, $-$ funktionieren ganz normal (komponentenweise).

Aufgabe 8.1*. Erstellen Sie eine 9×9 - `Matrix S` und belegen Sie n zufällig ausgewählte Elemente mit Zufallszahlen zwischen 1 und 9 $(0 \leq n \leq 81)$. Wir interpretieren das als teilausgefülltes Sudoku. Verwenden Sie dafür den Zufallszahlengenerator `rand`, und merken Sie sich die Indexpaare $(i, j) \in [1..9] \times [1..9]$, in denen bereits Einträge erfolgt sind, in einer Tabelle. (Initialisierung von `S: S:=Matrix(9,9)` initialisiert mit Nullen; diese interpretieren wir als leere Felder.)

Erstellen Sie eine Prozedur `checkS(S)`, die entweder `true` oder *errorinfo* zurückliefert, je nachdem ob das Sudoku korrekt ausgefüllt ist oder nicht. Dazu ist es sinnvoll, Hilfsprozeduren zu schreiben, die den entsprechenden Check für eine einzelne Zeile, eine einzelne Spalte oder einen einzelnen 3×3 - Block durchführen.

Der Ausgabewert *errinfo* soll spezifizieren, in welcher Zeile, welcher Spalte oder welchem Teilblock ein Mehrfachauftreten gefunden wurde; wählen Sie dafür ein geeignetes Format. (Die Prozedur bricht ab, sobald ein Fehler gefunden wird.)

Aufgabe 8.2*. Die Auswertung eines Polynomes

$$p(x) = a_0 + a_1\, x + \ldots + a_n\, x^n$$

an einer Stelle x führt man am besten mit Hilfe des *Horner-Schemas* durch, d.h., einer Schleife, die folgendes Rechenschema implementiert:

$$p(x) = a_0 + x\,(a_1 + x\,(a_2 + x\,(a_3 + \ldots)))$$

(die Schleife läuft 'von innen nach außen', beginnend mit $a_{n-1} + a_n\, x$).

Sei nun A eine gegebene quadratische Matrix (Datentyp `Matrix`) und v ein gegebener Vektor (Datentyp `Vector`). Verwenden Sie das Horner-Schema dazu, um ein Matrixpolynom an v auszuwerten, d.h. den Vektor

$$p(A)v = (a_0 + a_1\, A + a_2\, A^2 + \ldots + a_n\, A^n)\, v$$

zu berechnen. Dabei dürfen nur Matrix-Vektor-Multiplikationen verwendet werden. Implementieren Sie dies in einer Prozedur `matpol(A::Matrix,v::Vector,p::procedure)`, die A, v und eine Funktion `p` als Argument erwartet, die das Polynom repräsentiert. Falls es sich um kein Polynom handelt, liefert `degree` den logischen Wert `FAIL`. Verwenden Sie dies dazu, um in diesem Fall mit

`error` "This is not a polynomial";

abzubrechen. Überprüfen Sie auch, ob die Dimensionen korrekt sind, und brechen Sie ggf. mit einer entsprechenden Fehlermeldung ab. Berücksichtigen Sie den Fall des Nullpolynoms in korrekter Weise (mit Hilfe von `degree`, das in diesem Fall den Wert $-\infty$ zurückliefert).

Sie müssen also intern den Grad des Polynoms und die Koeffizienten a_i ermitteln. Verwenden Sie dazu `degree` und `coeff`.

Aufgabe 8.3*. Alles in Gleitpunktarithmetik (`evalf`):

Auch die Anwendung allgemeinerer, insbesondere analytischer Funktionen $f(x)$ auf quadratische Matrizen ist wohldefiniert. Sei z.B. $A \in \mathbb{R}^{n \times n}$ eine *diagonalisierbare* Matrix (einfachster Standardfall), d.h. $A = X\, \Lambda\, X^{-1}$ mit

X invertierbar und Λ diagonal (Diagonalelemente λ_i = Eigenwerte von A). Definieren Sie die Diagonalmatrix $f(\Lambda)$ mit Diagonalelementen $f(\lambda_i)$. Dann ist [1]

$$f(A) = X\, f(\Lambda)\, X^{-1}\,.$$

Führen Sie dies für die Funktion $f(x) = \exp(x)$ durch. Zum Test geben Sie X und Λ konkret numerisch vor; mittels `X^(-1)` erhalten Sie die Inverse. Vergleichen Sie mit `LinearAlgebra[MatrixExponential]` angewendet auf $A = X \cdot \Lambda \cdot X^{-1}$.

Aufgabe 8.4*. Funktionen, die über nichttriviale Prozeduren definiert sind (insbesondere wenn intern Fallunterscheidungen auftreten wie bei stückweise definierten Funktionen), können nicht unmittelbar differenziert oder integriert werden.

Auch das Plotten ist nicht ganz straightforward: Betrachten Sie z.B.

```
f := proc(x)
  if x<0 then
    return -1
  else
    return 1
  end if
end proc;
```

Dann funktioniert

```
plot(f(x),x=-1..1);
```

nicht (ausprobieren). Dann machen Sie es so:

```
plot('f(x)',x=-1..1);
```

und es funktioniert. Versuchen Sie diesen Effekt zu erklären.

Alternative: man verwendet `piecewise`. Studieren Sie die Dokumentation dazu und verwenden Sie es für die Definition einer stückweise definierten Funktion. Dann funktioniert auch die erste Variante. Auch `diff`, `int` funktionieren dann; dabei wird einfach abschnittsweise differenziert bzw. integriert. Testen Sie nun, ob ein *bestimmtes Integral* einer mittels `piecewise` definierten, stückweise stetigen Funktion mit Unstetigkeitsstellen (Sprungstellen) korrekt ausgewertet wird.

Zusatzfrage: Was fällt Ihnen in solchen Fällen beim plot auf? Bedenken Sie, dass `plot` auf einem adaptiven Auswertungsalgorithmus beruht, der Funktionswerte abtastet. `plot` 'weiß' aber z.B. nicht, ob tatsächlich Unstetigkeitsstellen vorliegen (das ist mittels endlich vieler Funktionsauswertungen nicht diagnostizierbar). Falls Ihnen der plot zu obigem Beispiel nicht gefällt, studieren Sie `?plot/discont` und probieren es aus.

Checken Sie auch folgendes Beispiel, mit Unendlichkeitsstellen:

`plot(tan(x),x=-5..5);` bzw. `plot(tan(x),x=-5..5,y=-4..4);`

und wiederholen Sie das Ganze auch mit `discont=true`.

Aufgabe 8.5*. Erstellen Sie eine Prozedur `comm(A::Matrix,B::Matrix)`, die den *Kommutator*

$$[A, B] := A\,B - B\,A$$

zweier quadratischer Matrizen zurückliefert. Bauen Sie Dimensionskontrolle ein (`error ...`).

Versuchen Sie nun die Identität

$$\begin{aligned}&(N_3\,A_3 + A_3\,M_3)\,A_2\,A_1 + A_3\,(N_2\,A_2 + A_2\,M_2)\,A_1 + A_3\,A_2\,(N_1\,A_1 + A_1\,M_1)\\ &= (N_3 + M_3 + N_2 + M_2 + N_1 + M_1)\,A_3\,A_2\,A_1 + [A_3, K_3]\,A_2\,A_1 + A_3\,[A_2, K_2]\,A_1 + A_3\,A_2\,[A_1, K_1]\end{aligned}$$

[1] Falls f ein Polynom ist, ergibt dies wieder genau $f(A)$ wie in Aufgabe 8.2, wie man leicht überlegt.

zu verifizieren, wobei

$$K_1 = M_1,$$
$$K_2 = M_1 + N_1 + M_2,$$
$$K_3 = M_1 + N_1 + M_2 + N_2 + M_3.$$

Hinweis: Definieren Sie zu diesem Zweck (für Dimension 2):

```
A[1] := Matrix(2,2,symbol='a')
```

usw., und probieren Sie aus, ob sich das ganze mittels `simplify` verifizieren lässt.[2] Ab Dimension 3 wird es bereits ein bisschen mühsam.

Didaktische Anmerkung dazu: Den *allgemeinen Beweis* der Identität (für beliebige Matrixdimensionen) kann man so nicht führen, selbst wenn man computergestützte Beweise im Prinzip akzeptiert. Das müsste man manuell machen, indem man die rechte Seite nach den Regeln der Matrixmultiplikation ausrechnet und zeigt, dass sie immer gleich der linken Seite ist (das ist ein kleines bisschen mühsam, aber eine reine Fleißaufgabe).

Sie können auch die (naheliegende) Vermutung aufstellen, welche analoge Identität wohl mit mehr als jeweils 3 Matrizen gelten wird, d.h. mit jeweils n Matrizen ($n \in \mathbb{N}$ beliebig). Der Beweis für allgemeines n ist dann natürlich aufwendiger. Wie so oft ist aber nicht der Beweis das Problem, sondern dass man die richtige Vermutung hat, d.h. dass man muss es erst 'entdecken'. Im Stadium des *Testens* von Vermutungen kann ein Computeralgebrasystem sehr hilfreiche Dienste leisten.

(Wozu die obige Identität gut sein soll: Das führt hier zu weit; es sei nur angemerkt, dass in gewissen Anwendungen das Verhalten des Ausdruckes erst nach der betreffenden Umformung *links* $\to$ *rechts* einer weiteren Analyse zugänglich ist.)

Aufgabe 8.6. Eine Familie von linearen Abbildungen $\varphi\colon \mathbb{R}^n \to \mathbb{R}^m$ sei als Prozedur `phi(x::Vector,m)` implementiert. Wenn ein Vektor der Länge[3] n übergeben wird, wird als Wert der Abbildung ein Vektor der Länge `m` zurückgegeben.

Definieren Sie irgendeine solche Familie von linearen Abbildungen als Prozedur.[4] Erstellen Sie sodann eine Prozedur `phimatrix(m,n)`, die für gegebenes `m,n` die Koeffizientenmatrix $A \in \mathbb{R}^{m\times n}$ der betreffenden Abbildung (bezüglich der kanonischen Basen im $\mathbb{R}^n, \mathbb{R}^m$) als `Matrix` zurückliefert.

Aufgabe 8.7. Erstellen Sie zwei Prozeduren `gradient(f,x,y,z)` und `Hessian(f,x,y,z)`, die eine zweimal stetig differenzierbare Funktion $f\colon \mathbb{R}^3 \to \mathbb{R}$ als Argument erwarten und den Gradienten

$$\operatorname{grad} f = \nabla f = \begin{pmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \\ \frac{\partial f}{\partial z} \end{pmatrix}$$

bzw. die Hesse-Matrix

$$H(f) = \begin{pmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x\,\partial y} & \frac{\partial^2 f}{\partial x\,\partial z} \\ \frac{\partial^2 f}{\partial x\,\partial y} & \frac{\partial^2 f}{\partial y^2} & \frac{\partial^2 f}{\partial y\,\partial z} \\ \frac{\partial^2 f}{\partial x\,\partial z} & \frac{\partial^2 f}{\partial y\,\partial z} & \frac{\partial^2 f}{\partial z^2} \end{pmatrix}$$

zurückliefern. Berücksichtigen Sie folgende beiden Fälle:

- Falls beim Aufruf nur ein Parameter angegeben wird (das ist zulässig), wird dieser als die Funktion f interpretiert, und ∇f (als `Vector`) bzw. $H(f)$ (als `Matrix`) werden zurückgegeben, ausgedrückt in den Variablennamen `x,y,z`.

---

[2] Möglicherweise nicht, weil `simplify` manchmal ein bisschen zickig ist. Stecken Sie da nicht zu viel Arbeit hinein.

[3] siehe `LinearAlgebra[Dimension]`

[4] Einfaches Beispiel: $\varphi(x)_k = \sum_{j=1}^{\min\{k,n\}} x_j$, $k = 1 \ldots m$, oder was immer Sie sich ausdenken. Es soll für beliebige $m, n \geq 1$ wohldefiniert sein.

- Falls beim Aufruf alle 4 Parameter angegeben werden, werden, nach Berechnung der Ableitungen, die für `x,y,z` konkret übergebenen Werte eingesetzt (mittels `subs`). (Diese können symbolisch sein, also andere Variablennamen, oder numerisch.)

Hinweis: Der Funktionsaufruf `nargs()` innerhalb einer Prozedur liefert die Anzahl der konkret übergebenen Argumente zurück. Zulässig sind hier die Werte 1 und 4. (Weitere Überprüfungen brauchen Sie hier nicht einzubauen.)

Aber *Achtung:* Jeder Parameter, der intern in irgendeiner Form verwendet wird, muss entweder tatsächlich angegeben oder mit einem *Default-Wert* versehen werden. Machen Sie es daher z.B. so:

```
gradient:=proc(f,my_x:='x',my_y_:='y',my_z:='z') # definiert Default-Werte
local x,y,z: # oder irgendwelche andere Variablennamen
...
```

und arbeiten Sie intern beim Differenzieren mit den lokalen Variablen `x,y,z`. Zum Schluss setzen Sie für `x,y,z` die Werte `my_x,my_y,my_z` ein. Warum so kompliziert? Der Grund dafür: Eine Deklaration der Gestalt

```
gradient:=proc(f,x:='x',y:='y',z:='z')
```

wird nicht akzeptiert.

Die Aufrufe `gradient(f)` und `gradient(f,'x','y','z')` sind dann äquivalent (analog für `Hessian`).

Aufgabe 8.8. Jemand gibt Ihnen eine Prozedur `phi(x::Vector)`, die eine Funktion $\varphi : \mathbb{R}^3 \to \mathbb{R}$ implementiert. Sie wollen nun feststellen, ob φ eine *lineare* Funktion ist, d.h. von der Gestalt $\varphi(x) = a_1\,x_1 + a_2\,x_2 + a_3\,x_3$. Erstellen Sie eine Prozedur `testlin(phi)`, die das überprüft, und geben Sie die so ermittelte Information in geeigneter Weise zurück.

Anmerkung: Falls `phi` intern sehr kompliziert gebaut ist, kann es sein, dass der Linearitätstest scheitert, z.B. weil `simplify` überfordert ist. Die (vorsichtige) Diagnose beim Scheitern des Linearitätstests (in Worten ausgedrückt) wäre daher: 'Linearität konnte nicht verifiziert werden' anstelle von 'Funktion ist nichtlinear'.

Aufgabe 8.9. Ein differenzierbares Vektorfeld $F\colon \mathbb{R}^2 \to \mathbb{R}^3$ definiert eine glatte Fläche im $\mathbb{R}^3$: Die Punkte der Fläche sind gegeben durch $(x,y,z) = F(u,v) = (F_1(u,v), F_2(u,v), F_3(u,v))$, wobei $(u,v) \in G \subset \mathbb{R}^2$. Man spricht auch von einer *Parameterdarstellung* der Fläche (siehe Beispiel unten).[5]

Sei z.B. $G = [a,b] \times [c,d]$ ein Rechteck in der (u,v)-Ebene. Dann ist der Flächeninhalt der betreffenden Fläche gegeben durch das Doppelintegral

$$\int_F dS := \int_{v=c}^{d} \int_{u=a}^{b} \sqrt{m(u,v)}\; du\, dv, \qquad \text{mit} \quad m(u,v) := \det \begin{pmatrix} \frac{\partial F}{\partial u} \cdot \frac{\partial F}{\partial u} & \frac{\partial F}{\partial u} \cdot \frac{\partial F}{\partial v} \\ \frac{\partial F}{\partial u} \cdot \frac{\partial F}{\partial v} & \frac{\partial F}{\partial v} \cdot \frac{\partial F}{\partial v} \end{pmatrix}.$$

Dabei ist

$$\frac{\partial F}{\partial u} = \begin{pmatrix} \frac{\partial F_1}{\partial u} \\ \frac{\partial F_2}{\partial u} \\ \frac{\partial F_3}{\partial u} \end{pmatrix}$$

(analog für v), und $\cdot$ bezeichnet das innere Produkt zweier Vektoren.

Verwenden Sie diese Formel, um mit Hilfe von Maple die Oberfläche der Einheitskugel im $\mathbb{R}^3$ zu berechnen. Ihre Parametrisierung in Kugelkoordinaten (u,v) lautet

$$F(u,v) = \begin{pmatrix} \sin u\, \cos v \\ \sin u\, \sin v \\ \cos u \end{pmatrix}, \quad (u,v) \in [0,\pi] \times [0,2\pi]\,.$$

[5] Einfachster Fall: $u = x$, $v = y$; $F(x,y) = (x,y,z(x,y))$ mit der Höhenkoordinate $z(x,y)$. Aber z.B. eine Kugel kann man so nicht komplett parametrisieren, weil $z(x,y)$ zwei verschiedene Werte annehmen müsste (oben/unten).

Aufgabe 8.10. Verwenden Sie `plot3d`, um ein Bild der Einheitskugel zu zeichnen (Parameterdarstellung wie in Aufgabe 8.9 angegeben).

Andere nette Tests für `plot3d` sind das *Möbiusband* (eine Fläche mit nur einer Seite) oder die *Klein'sche Flasche*, (eine geschlossene Fläche mit nur einer Seite, die sich selbst durchdringt). Die betreffenden Parameterdarstellungen finden Sie z.B. auf Wikipedia.

Anmerkung: Der optionale Parameter `coords` in `plot3d` hat nichts mit der gewählten Parameterdarstellung zu tun. Vielmehr: Wenn man z.B. `coords=spherical` angibt, dann werden die 3 Komponenten F_1, F_2, F_3 des Vektorfeldes, das die Fläche beschreibt, nicht als kartesische Koordinaten (x, y, z), sondern als dreidimensionale Kugelkoordinaten eines Flächenpunktes im $\mathbb{R}^3$ interpretiert (mit dem Abstand vom Ursprung als weitere Koordinate). (Siehe `? plot3d/coords`.) Dennoch werden diese Komponenten von `plot3d` in einem kartesischen Koordinatensystem dargestellt. Gibt man beim Kugelbeispiel `coords=spherical` an, so erhält man als Bild also keine Kugel, sondern etwas anderes Lustiges. Das Möbiusband mit `coords=cylindrical` sieht auch ganz nett aus.