Winfried Auzinger
Dirk Praetorius

Sommersemester 2013
17./18. April 2013

Übungsaufgaben zur VU Computermathematik

Serie 4

Generelle Anmerkung zu den Maple-Übungen:

Die Aufgabenstellungen sind in englischer Sprache formuliert. (Nebeneffekt: ein bisschen gewöhnen an die englische Fachsprache.) Nehmen Sie sich die Zeit, die Aufgabenstellungen genau durchzulesen; diese enthalten auch Hintergrundinformationen über das jeweilige Thema und können daher gelegentlich etwas ausführlicher geraten. Viele der Übungsaufgaben sind keine reinen 'Maple-Aufgaben', sondern enthalten auch eine mathematische Problemstellung, die Sie, ggf. mit entsprechenden Hinweisen, zunächst verstehen bzw. knacken müssen. Manche andere wieder sind experimenteller Natur (was für den Physiker das Labor ist, ist für den Mathematiker der Computer). Und bedenken Sie: Der Name der LVA ist Computer*mathematik*.

Für vielen Fragestellungen gibt es innerhalb von Maple schon fertige Lösungen Es spricht aber nichts dagegen, so etwas als Übungsaufgabe zu verwenden (auch in anderen Übungen berechnen oder beweisen Sie Dinge, die schon andere vor Ihnen berechnet bzw. bewiesen haben).

Es kommt gelegentlich vor, dass Sie einen Befehl benötigen, der in der Vorlesung (noch) nicht besprochen wurde Das ist kein Drama; für derartige Fälle werden Hinweise gegeben, manchmal einfach nur das richtige Stichwort, für das Sie Details in der Hilfe nachschlagen können. Lesen Sie die Hinweise genau durch; manches müssen Sie ggf. noch selbst herausfinden. Orientieren Sie sich auch mit Hilfe des Flyers (siehe Homepage). Manche der Aufgaben (ohne Stern) haben auch den Zweck, dass Sie sich einen in der Vorlesung (aus Zeitgründen) nicht oder noch nicht im Detail besprochenen Stoff aktiv anhand von Beispielen selbst erarbeiten. Nützen Sie generell die Maple-Hilfe systematisch – für die praktische Arbeit ist dies unumgänglich.

Ihre Codes sollten Sie so weit wie möglich auf Korrektheit testen. Dokumentieren Sie Ihre Worksheets auch in angemessener Weise mittels Zwischentexten bzw. Kommentaren (`#`). Das Hauptziel dabei sollte immer sein, dass Sie selbst später noch erkennen können, was sie sich dabei gedacht haben.

Exercise 4.1.

a) Design a function `digit(n,k)` which expects a natural number `n` $\in \mathbb{N}$ as argument and returns its `k`-th decimal digit. E.g.,

```
digit(1034,1)  -->   4
digit(1034,2)  -->   3
digit(1034,3)  -->   0
digit(1034,4)  -->   1
digit(1034,5)  -->   0
...
```

Hint: Use `mod`.

b) Design a function `length(n)` which expects a natural number `n` $\in \mathbb{N}$ as argument and returns its 'length', i.e., the number of decimal digits of `n`. E.g., `1034` has 'length' 4.

Hint: Use the decimal logarithm `log[10](.)` and the rounding function `floor(.)` (rounding downwards to nearest integer).

c) Use **a)** and **b)** to design a function `sumdigits(n)` which expects a natural number `n` $\in \mathbb{N}$ as argument and returns the sum of its decimal digits.

Make sure that special cases like 1, 10, 100, 1000, ... are treated correctly.

Exercise 4.2.

a) Design a function `dotproduct(a,b)` which expects two lists `a` and `b` of arbitrary lengths, interprets them as real vectors and returns the inner product

$$a \cdot b := \sum_i a_i \, b_i$$

of these vectors. Use `add`. In order to handle the case where two lists of different length are passed to your function, use `min(.)` and `nops(.)` to cut down the the shorter length. Check what happens if you evaluate `dotproduct(.,.)` with numeric, symbolic, or mixed data.

Furthermore, design a function `norm(a)` with returns the length, or *Euclidean norm* of the vector a,

$$\|a\|_2 := \sqrt{a \cdot a}$$

b) Design a function `isorthogonal(a,b)` which returns `true` if the lists `a` and `b` represent orthogonal vectors, and `false` otherwise. Use `simplify`, `evalb`, and your function `dotproduct(a,b)` from **a)**. Check what happens if you evaluate `isorthogonal(.,.)` with numeric, symbolic, or mixed data.

c) Design a function `isparallel(a,b)` which returns `true` if the lists `a` and `b` represent parallel vectors, and `false` otherwise. Use `simplify`, `add`, `evalb`, and your function `dotproduct(a,b)` from **a)**.

Hint: In any case, the *Cauchy-Schwarz inequality*

$$(a \cdot b)^2 \;\le\; \|a\|_2^2 \cdot \|b\|_2^2, \quad \text{i.e.,} \quad \Big(\sum_i a_i \, b_i\Big)^2 \;\le\; \Big(\sum_i a_i^2\Big) \cdot \Big(\sum_i b_i^2\Big)$$

is valid, and equality (=) holds true 'iff' (i.e., if and only if) the vectors a and b are parallel.

Check what happens if you evaluate `isparallel(.,.)` with numeric, symbolic, or mixed data.

Exercise 4.3. Let $n \in \mathbb{N}$ be given and $p(x) = c_0 + c_1\, x + \ldots + c_n\, x_n$ be an arbitrary polynomial of degree $\le n$. We represent it by a list `p` of length $n+1$ containing its coefficients a_j. (Warning: the first index in a list is always 1, so you have to reorganize your indices.

a) Design a function `evaluate(p,x)` which evaluates $p(x)$.

b) Design a function `antiderivative(p)` which returns the coefficients of the antiderivative (*Stammfunktion*) of $p(x)$ again in the same format, adding a constant of integration `C`.

c) Design a function `definite_integral(p,a,b)` which computes $\int_a^b p(x)\,dx$ using `antiderivative`.

Hint: You may compare with the results provided by the built-in integrator `int`.

Exercise 4.4. Let `x = [0,1/4,1/2,3/4,1]` represent an equidistant subdivision of the interval $[0,1]$. Determine the coefficients ω_i, $i = 1 \ldots 5$, such that

$$\int_0^1 p(x)\,dx \;=\; \sum_{i=1}^{5} \omega_i \, p(x_i) \tag{1}$$

for all polynomials $p(x)$ of degree ≤ 4.

Hint: Use `solve` to solve a system of linear equations which results from the requirement that the monomials functions $p(x) = 1, x, x^2, x^3, x^4$ are exactly integrated by the weighted sum in (1). Check your result, e.g., using `int`, or your function `definite_integral` from 4.3.

Exercise 4.5. Assume that `A` und `B` are finite sets. Define functions which expect sets a arguments and perform the following operations:

- $(\texttt{A},\texttt{B}) \mapsto (\texttt{A} \cup \texttt{B}) \setminus (\texttt{A} \cap \texttt{B})$
- $(\texttt{A},\texttt{B},\texttt{x}) \mapsto$ `true`, if `x` is contained in *exactly one* of the sets, and `false` otherwise. Use `xor` and `evalb`.
- $(\texttt{A},\texttt{B}) \mapsto \texttt{A} \times \texttt{B}$ (Cartesian product), a set of ordered pairs which you can represent by lists `[a,b]` of length 2.

  Hint: Use `seq(...)` in a nested ('*geschachtelt*') way.

Exercise 4.6. An *equivalence relation* between elements of a set X is a formally defined as a subset E of $X \times X$ with the properties

- *reflexivity:* $(x,x) \in E$ for all $x \in X$,
- *symmetry:* $(x,y) \in E \;\Leftrightarrow\; (y,x) \in E$,
- *transitivity:* $(x,y) \in E$ and $(y,z) \in E \;\Rightarrow\; (x,z) \in E$.

Relation $(x,y) \in E$ is usually written as $x \sim y$. It is easy to prove that each equivalence relation induces a decomposition of X into a collection of pairwise disjoint *equivalence classes* X_i with $\bigcup_i X_i = X$. Each of the X_i consists of sets of elements equivalent to each other.

Conversely, specifying a collection of pairwise disjoint sets (equivalence classes) X_i defines an equivalence relation on $X = \bigcup_i X_i =$ via

$$x \sim y \;:\Leftrightarrow\; x,y \in X_i \text{ for some } X_i\,.$$

a) Design a function `iseqclasses(C)` which expects a set `C` consisting of sets of numbers as argument, e.g.,

```
{{1,2},{3,4},{5,6,7}}
```

Your function checks whether these sets are pairwise disjoint and returns `true` in this case, and `false` otherwise. I.e., `true` means that the elements of `C` define equivalence classes X_i on the union X of their elements.

Hint: Use a nested construction, applying `evalb, seq, intersect`, and `minus`. It is also convenient to define a function `isempty(.)` which decides whether a set is empty (such a function is not built-in to Maple).

b) Design a function `isequivalent(x,y,C)` which returns `true` if $\texttt{x} \sim \texttt{y}$ in the sense of **a)**, and `false` otherwise. Here it is assumed that `C` is valid set of equivalence classes in the sense of **a)**.

Exercise 4.7. Provide a computer-assisted proof of *Young's inequality*

$$x\,y \;\le\; \frac{x^p}{p} + \frac{y^q}{q} \quad \text{for all} \quad x,y \ge 0\,, \tag{2}$$

where $p > 1$, and q is the 'conjugate' of p, satisfying $\frac{1}{p} + \frac{1}{q} = 1$.

Proceed as follows: First, declare

```
assume(x>=0); assume(y>=0); assume(p>1);
```

which means that you tell Maple that it should respect these variable properties (x cannot be negative, etc.). Express q in terms of p and define the function

$$f(x) \;:=\; \frac{x^p}{p} + \frac{y^q}{q} - x\,y$$

Evidently, the function f satisfies $f(0) > 0$ and $f(x) \to \infty$ for $x \to \infty$. Now, use `diff(f(x),x)` to compute the derivative $f'(x)$, and solve equation $f'(x) = 0$ for x. Evaluate $f(x)$ at this point and simplify. Finally, argue that the outcome implies Young's inequality (2).

Remark: Of course, you can also perform the required computations by hand, but Maple can do the work for you. However, Maple cannot tell you how to *get the idea* behind such a proof.

Exercise 4.8. Design the functions `ip3 := (x,y) -> ...` and `op3 := (x,y) -> ...`, which expect lists `x,y` of length 3 as arguments and interpret them as vectors in $\mathbb{R}^3$. `ip3` and `op3` return the inner product $x \cdot y$ and the outer product $x \times y$ of these vectors, respectively, where `op3` again returns a list of length 3.

Use these functions to verify the relations $a \times b \perp a,\ a \times b \perp b$, and the identities

$$\|a \times b\|_2^2 = \|a\|_2^2 \, \|b\|_2^2 - (a \cdot b)^2 \quad \text{(Lagrange's identity)},$$
$$a \times (b \times c) = (a \cdot c)\, b - (a \cdot b)\, c\,,$$
$$(a \times b) \cdot (c \times d) = (a \cdot c)(b \cdot d) - (b \cdot c)(a \cdot d)\,,$$

by passing 'generic', symbolic data (lists) `a,b,c,d` to `ip3` and `op3`. Use `simplify` (this may not always do the desired job), or `expand`. You may also first test with numerical (integer) data.

Hints:

- $\|x\|_2$ is the length (Euclidean norm) of x, i.e., $\|x\|_2^2 = x \cdot x$.
- Multiplication of a list with a scalar returns the componentwise product as a list.
- For 'generic', symbolic data, simply set `a:=[a1,a2,a3]`, or using variables with indices (output looks nicer), e.g.

```
a := [alpha[1],alpha[2],alpha[3]]
```
