Winfried Auzinger
Dirk Praetorius

Sommersemester 2015
22.–24.04.2015

Übungsaufgaben zur VU Computermathematik

Serie 5

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 oft 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 sollen. 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 Computermathematik.

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 immer wieder vor, dass Sie etwas benötigen, das in der Vorlesung (noch) nicht besprochen wurde 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 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, z.B. was die Erstellung von Grafiken, Animationen etc. betrifft.

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 Minimalziel dabei sollte immer sein, dass Sie selbst später noch erkennen können, was sie sich dabei gedacht haben. [1]

Aufgaben mit (∗) (kommt manchmal vor) sind ein wenig anspruchsvoller.

Exercise 5.1. *Basic syntax.*

a) Design a function `firstprimes(n)` which returns a list containing the first `n` prime numbers.

Hint: Use `seq` and `ithprime`.

b) *The 'from-to' operator `..` is used to represent a range of integers.* [2]

Design a function `rangetolist(intrange)` which expects an expression `intrange` of the type `..`, i.e., of the form `m..n` as its argument and converts it to the list

```
[m,m+1,...n]
```

Also test what happens if `m` is larger than `n`.

Hint: This readily works using `seq`.

c) Design a function `isinrange(k,intrange)` which expects an integer `k` and a range `intrange` (see **b)**) as its arguments and returns `true` if `k` is contained in this range and `false` otherwise.

Hint: Use logical operations. Furthermore, you need the function `op(`*i*`,`*expr*`)` which selects the *i*-th operand from an expression *expr*. Check how to use `op` for extracting `m` and `n` from a an expression `m..n` of type `..` (it works in a rather natural way).

---

[1] *Documentation is like sex: If it's good, it's very good. If it's bad, it's better than nothing.*

[2] Remark: `..` is also used to represent real intervals, e.g., when specifying the lower and upper limits of a definite integral.

Exercise 5.2. *Binomial coefficients.*

a) Design a function `my_binomi(a,b,n)` which returns the expression [3]

$$\sum_{k=0}^{n} \binom{n}{k} a^k \, b^{n-k}$$

Test your function with different data (with symbolic and/or numerical values for `a,b,n`). What happens if you `factor` the result, e.g. of `my_binomi(a,b,10)` ?

Hint: Use `add` and `binomial`.

b) Implement your own version of the binomial coefficient $\binom{n}{k}$ in form of a function `my_binomial(n,k)`, which expects two nonnegative numerical integer values as its arguments. Do not use factorials but use `mul` and minimize the number of (integer) multiplications. To this end you need to distinct between two cases – use `` `if`(...) `` for this purpose.

Example for using `` `if` ``: The function

```
x -> `if`(x=0,0,1)
```

returns 0 for $x = 0$ and 1 otherwise.

c) For arbitrary $c \in \mathbb{R}$ and $k \in \mathbb{N}_0$, the (generalized) binomial coefficient is defined as

$$\binom{c}{k} := \frac{c\,(c-1)\cdots(c-k+1)}{k!}$$

Implement this in form of a function `my_generalized_binomial(c,k)`. Compare your results with `binomial`.

Hint: Use `mul`.

Exercise 5.3. *Symbolic summation.*

a) Use `sum` to evaluate the sums

$$\sum_{k=1}^{n} k, \quad \sum_{k=1}^{n} k^2, \quad \sum_{k=1}^{n} k^3, \ \ldots \qquad \sum_{k=1}^{n} \frac{1}{k}, \quad \sum_{k=1}^{n} \frac{1}{k^2}, \ \ldots$$

$$\sum_{k=1}^{n} x^k, \quad \sum_{k=1}^{n} k\,x^k, \quad \sum_{k=1}^{n} k^2\,x^k, \quad \ldots, \quad \sum_{k=1}^{n} k^p\,x^k$$

Here `n,x,p` are symbolic objects (unassigned, i.e., no value assigned). Check what happens, and try to convert the result into a form as simple as possible.

b) For **a)** you have used the syntax `sum(...,k=1..n)`. Do the same again using

```
sum(...,k)
```

and check/interpret the outcome.

c) Repeat **a)** and sum up to infinity,

```
sum(...,k=1..infinity)
```

and check/interpret the outcome. What behavior do you observe for the (generalized) geometric series $\sum_{k=1}^{\infty} k^p\,x^k$?

d) Check what happens if you enter (using `binomial`)

$$\sum_{k=0}^{\infty} \binom{c}{k} x^k$$

Hint: This called the *binomial series* $(c, x \in \mathbb{R})$. (Do you know for what values of x this series converges?) In what way does this series specialize if $c \in \mathbb{N}_0$?

Furthermore, replace `binomial` by your function `my_generalized_binomial` from **5.2 c)**. What happens? Fix the problem.

---

[3] Binomial theorem, due to *Alessandro Luigi Franceso Binomi* (1485–1543)

Exercise 5.4. *Set operations.*

a) Design a function `equivalent(A,B)` which expects to sets `A,B` as its arguments and returns `true` if these have the same number of elements, otherwise `false`.

b) Design two functions `add_element(element,A)` and `remove_element(element,A)` which expects any object `element` and a set `A` as its arguments and returns a new set with `element` added to or removed from `A`, respectively.

Hint: Use standard set operations.

c) Design a function `cartprod(A,B)` which expects to sets `A,B` as its arguments and returns their cartesian product $A \times B$ in form of a set consisting of lists of length 2.

d) (∗) Design a function `checktype(A,t)` which expects a set `A` and a type name `t` (e.g., `symbol, numeric, integer, ...`) as its arguments and returns `true` if all elements of the set have the same type specified by `t`, otherwise `false`.

Hint: Use `seq`, `evalb`, and use `is(`*element,type*`)` to check if *element* has the specified *type*. With a control structure (a `for`-loop) this is rather easy to realize, but you cannot use a `for`-loop within a function. Therefore you need some workaround. Experimental!

Exercise 5.5. *Recursive functions.*

a) Consider the recursive function

```
f:= n->`if`(n=0,1,n*f(n-1))
```
(`n` assumed to be a nonnegative integer)

What does it represent? Also, check what happens if you call it with a negative argument.

b) Design a recursive function `rbinomial(n,k)` which computes the binomial coefficient $\binom{n}{k}$ $(n, k \in \mathbb{N}_0)$ in a recursive way according to the pattern (recursion with respect to `k` for `n` fixed)

$$\binom{n}{k} = \underbrace{\underbrace{\underbrace{1}_{k=0} \cdot \frac{n}{1}}_{k=1} \cdot \frac{n-1}{2}}_{k=2} \cdot \frac{n-2}{3} \quad \cdots}_{k=3}$$

For testing, compare with `binomial`. Furthermore, check what happens if you call `[r]binomial` with `k` and/or `n` not numerically specified. E.g., check the outcome of

```
> n := 'n';  k := 3;
> binomial(n,k);
> rbinomial(n,k);
```

Exercise 5.6. *Basic plots.*

Besides the normal `plot` command for real functions, there are many more commands for plotting functions or discrete data. (In particular, the package `plots` contains many special versions.) Consult the help pages!

a) Use `plot` to plot a real functions of your choice. Check the help page and try to generate a 'nice' plot by adjusting parameters. E.g., modifying the default values of the parameters `color`, `thickness`, `axes` (and many others) may be useful.

b) A discrete analog of `plot` is `plots[listplot]` [4] contained in the package `plots`. Use `listplot` to visualize the discrete function $f\colon \mathbb{N} \to \mathbb{N}$, defined by

$f(n) := 1$ if n is prime, and 0 otherwise

Hint: Use `` `if`(...) `` and `is(...,prime)`.

Here, e.g., modifying the default values for the parameters `symbol`, `symbolsize`, and `style` (among others) may be particularly useful in order to produce a nice-looking listplot.

[4] This syntax calls `listplot` without loading the complete contents of the `plots` package into memory. See also **5.7 b)**, (ii).

Exercise 5.7. *A bit of analytic geometry; a system of polynomial equations; implicit plots.*

a) Design a function `isline(A,B,C)` which expects 3 lists `A=[`a_1, a_2`]`, `B=[`b_1, b_2`]`, and `C=[`c_1, c_2`]` with rational entries as its arguments. `A,B,C` represent Cartesian coordinates of points in the plane. The function `isline` returns `true` if these points are located on a common line, otherwise `false`.

Hint: There are different possibilities to check this. (For instance, one may use a 2×2 determinant; use `evalb` to check whether this determinant is [non]zero.)

b) Two curves in the plane are given in an implicit way by the equations

$$\begin{aligned} x^2 + y^2 &= 1 \\ 2\,x^3 + 2\,y^2 &= 1 \end{aligned} \tag{5.7b}$$

(i) $(*)$ Find all intersection points of these curves.

Guideline: Use `solve`. As an answer you get a list of so-called `RootOf` expressions.

Here, `Rootof(`*equation*`)` represents (in an implicit way) all possible solutions of *equation*. For the present example, these `RootOf` expressions can be exactly evaluated: Use

`allvalues(...)` and `evalf(...)`

to obtain explicit numerical values. From these you can read off[5] the coordinates of the intersection points of the given curves.

(ii) Plot the given curves in the following way:

```
> with(plots); # activate package plots with special plotting commands
               # (implicitplot, display, listplot, and many others)
> implicitplot(x^2+y^2=1,x=-2..2,y=-2..2); # implicit plot with window specified
> implicitplot(x^2+y^2=1,x=-2..2,y=-2..2); #
> plotstruct1 := implicitplot(x^2+y^2=1,x=-2..2,y=-2..2):     # generate plot structure
> plotstruct2 := implicitplot(2*x^3+2*y^2=1,x=-2..2,y=-2..2): #
> display(plotstruct1,plotstruct2); # render plot structures in a single plot
```

Here, `implicitplot` plots a curve which is given in an implicit way by an equation, as in (5.7b). If you assign a plot to a variable (as in `plotstruct1 := ...`), the plot data are stored (in an internal format) and assigned to the variable. With `display` you can render several plots simultaneously.

Exercise 5.8. *A function expecting another function as its argument. A function returning another function as its result.*

a) Design a function `fmean(f,X)` which expects a function `f(x)` and a list `X` of values x_i as its arguments and returns the value

$$\frac{1}{n}\sum_{i=1}^{n} f(x_i) \quad (n \text{ is the number of entries in } \texttt{X})$$

b) Design a function `fline(P1,P2)` which expects two lists `P1=[x1,y1]` and `P2=[x2,y2]` as its arguments and returns a function whose graph is the straight line connecting `P1` and `P2`. If this is not well-defined, return a function which in turn returns `NULL`[6] as its function value.

Test `fline` with symbolic as well as numeric data `P1,P2` and with data of mixed types.

Hint: Use '`if`'.

---

[5] Observe that some of the solutions are not real-valued.

[6] `NULL` is a pre-defined constant representing 'nothing'.