Winfried Auzinger
Dirk Praetorius

Sommersemester 2016
10. Mai 2016

Übungsaufgaben zur VU Computermathematik

Serie 9

Eine Kollektion verschiedener Problemstellungen, teilweise mit stofflichen Ergänzungen, entsprechend erläutert.

Exercise 9.1: *Exception handling. A real* `sqrt` *implementation.*

The `try ... end try` construct allows you to protect critical parts of your code, with a controlled error handling by the `catch` branch if the `try` branch fails.[1]

Example:

```
try
   b := 1/a;
   c := 1/d;
catch:
   error "Division by zero!"
   # maybe some alternative computation here...
finally # (optional; if applicable)
   print("The finally block is executed in any case.")
end try:
```

A simple application:

When `sqrt(...)` is called with a negative real argument you get an imaginary answer. Here we aim for a real implementation which results into an error message in this case.

a) *Provide such an alternative implementation of* `sqrt` *by means of a procedure* `rsqrt(...)`*, and test it on the function f from Ex.* **6.2**.

Hint: There is some odd behavior (at least in Maple 2015.1):

See what happens when you call f from **6.2** *with an integer argument such that f is not well-defined. What happens? Remedy: In your procedure* `rsqrt`*, replace* `sqrt(...)` *by* `evalf(sqrt(...))`*. Test.*

b) *Use your procedure* `rsqrt` *within a* `try`*-block, catching negative arguments.*

Exercise 9.2: *Some animations. Further tools from the* `plots` *package.*

a) The function `animate` can be used to produce videos, i.e., a sequence of plots depending on a parameter. After defining the corresponding plot structure, rendering of the animation is performed in an interactive way using a context menu.

Consult `?animate` *(look at the examples). Use animate to visualize the behavior of the Taylor polynomials $1 + x + \frac{x^2}{2} + \ldots + \frac{x^n}{n!}$ for $x \in [0,3]$ and $n = 1, 2, \ldots$ (n is the parameter for the animation). Also, use an example of your own choice.*

---

[1] Such a mechanism is implemented in many programming languages. This is often more useful than trying to avoid in advance, by various `if` constructs, that an error occurs, in particular if there are several 'critical operations' or if you do not exactly know where an exception may occur.

b) An animation of the evolution of a function can be generated using `animatecurve`.

Show some nice example.

c) Another way of generating a video is to produce several plot structures and to display them in an animated way, using `display` with option `insequence=true`.

Choose a function $f(t)$ and generate plots on intervals $[0,T]$ with increasing values for T. Use the `plot` *option* `filled=true`*. Then, use* `display` *with option* `insequence=true` *to animate these plots.*

This provides a visualization of the behavior of $\int_0^T f(t)\,dt$ with increasing T.

d) *Also check* `animate3d` *and show some nice example.*

Remark: You can export your video in form of an animated `.gif` file.

Exercise 9.3: ***Solving a system of polynomial equations.***

Consider the system of three polynomial equations

$$\begin{aligned} x_1 + x_2 + x_3 &= 1 \\ 4\,x_2\,x_3 + 2\,x_3^2 &= 1 \\ 3\,x_2^2\,x_3 + 9\,x_2\,x_3^2 + 3\,x_3^3 &= 1 \end{aligned}$$

in the three variables x_1, x_2, x_3. We represent the system in form of a list containing these three equations.

a) *Solve this system.* You will get an answer in terms of an expression of the form

```
RootOf(12*_Z^4-24*_Z^2+16*_Z-3)
```

b) *What does this* `RootOf` *expression represent? Use* `allvalues` *and* `evalf` *to find numerical values for all real solutions, and verify that they are correct.*

Exercise 9.4: ***A property of quadratic equations, investigated experimentally.***

Consider the quadratic equation

$$z^2 + b\,z + c = 0, \qquad \text{with} \quad b > 0 \quad \text{and} \quad c > 0.$$

a) It can be shown that both solutions $z_{1,2}$ of such an equation have negative real parts. Here we do not try to prove this, but we check it by experiment.

To verify the above assertion, use `plot3d` to plot the real part of both solution over some range of values $b > 0$ and $c > 0$.

b) Make an analogous plot for the imaginary parts and explain the outcome.

c) Let $c = 1$. Use `plots[complexplot]` to visualize the movement of both solutions of the quadratic equation in the complex plane when the parameter b varies (i.e., starts at $b = 0$ and increases).

Furthermore, use `plots[display]` with option `insequence=true` to produce a video of this movement (actually, these are two curves in the complex plane).

Exercise 9.5: ***Formatted input.***

Assume that the coefficients of a multivariate polynomial expression are encoded in a text file in a way as shown here (this example refers to six variables $x_1, \ldots, x_6$):

```
[0,2,0,1,0,1]  7
[0,1,1,1,1,0]  6
[0,0,2,1,0,0] -2
[2,0,0,0,0,0]  3
[0,0,0,0,0,0] -1
```

Each of the lines represents a power product, where the entries in the list specify the powers with which the variables $x_1, \ldots, x_6$ occur, and the number at the end of the line specifies a multiplicative factor. I.e., this text file represents the expression

$$7\,x_2^2\,x_4\,x_6 + 6\,x_2\,x_3\,x_4\,x_5 - 2\,x_3^2\,x_4 + 3\,x_1^2 - 1\,.$$

• *Design a procedure* `readmultinom(`*filename,var*`)` *which reads the data from such a file and returns the corresponding multinomial expression. var is the variable name (e.g., var=*`x`*).*

Hint: Use `readline` followed by `sscanf`. Note that with the `%a` format specifier, a list is scanned as a single object. For the coefficient at the end of the line, use `%d`. You may assume that the format is correct, in particular, that all lists have the same length (which you have to determine in a first step, when scanning the first line).

Test with the above example and also another one.

Exercise 9.6: ***Scanning a multinomial expression.***

A converse of Ex. **9.5**:

Extracting the coefficients and exponents from a given multinomial expression (e.g., for saving them to a text file) is a slightly more complicated operation. You need some understanding about its internal representation.

We may proceed as follows: At first we only consider a single term of the form

$c\,x_1^{p_1} \cdots x_n^{p_n}$, e.g., `2*x[1]*x[3]^2`.

We assume that n, the number of variables `x[1]`,`x[2]`,... involved, is a priori known. Our job is to extract c and $p_1, \ldots, p_n$. We may realize this in the following way:

- Use `subs(...)` to replace all variables $x_1, \ldots, x_n$ by 1. This results in the value c.
- Use `degree(...)` with respect to all the variables x_j in order to find the exponents p_j (which may be also be zero).

a) *Realize this in form of a procedure which returns the expression* `[`$p_1, \ldots, p_n$`]`, c.

b) *In order to learn how to handle a sum of such terms (i.e., a general multinomial expression) m, find out by considering some examples what you get from* `op(1,`*m*`)`, `op(2,`*m*`)`,...,`op(nops(`*m*`),`*m*`)`. *Check how minus signs are handled. What is* `op(0,m)` *?*

Exercise 9.7: ***Formatted output.***

Assume that a system of multinomial expressions is given in form of a list of such expressions, e.g., as in Ex. **9.3**.

• *Store the data of the system to text files*[2] *in the same format as in Ex.* **9.5**. *Each expression is to be stored on a separate file, e.g.,* `sys01.txt`, `sys02.txt`, *. . . .*

Exercise 9.8: ***Using*** `unapply` ***(see lecture notes, part II); using*** `.mpl` ***files***

As an example consider a given rational function $r(x)$, e.g., $r(x) = \frac{x}{x^2-x+1}$. Assume we need some of its higher derivatives for repeated use in a numerical algorithm. Then it will be inefficient to perform symbolic differentiation again and again. Rather, we will statically store the 'ready-cooked' formula for such a derivative, e.g.

$$\frac{d^5}{dx^5}\,\frac{x}{x^2-x+1} = \frac{120\,(x^6 - 15\,x^4 + 20\,x^3 - 6\,x + 1)}{(x^2-x+1)^6}$$

a) *Let the function* `r(x)` *represent the above (or some other) rational function. Use* `diff`, `normal`, *and* `unapply` *to define a function* `r5(x)` *which returns the expression for the 5 - th derivative.*

[2] This will be useful, for instance, if the system is to be exported to external software like MATLAB.

b) You can now use this function in your worksheet. But assume you want to use it in other worksheets. Of course you may copy and paste the code, but a more flexible way ist to store it in a text file *filename*`.mpl`. Example:

```
save(r,r5,"rat.mpl")
```

Try this and look at the textfile `rat.mpl`*. Then,* `restart` *your worksheet and perform*

`read("rat.mpl")` or `read("rat.mpl"):`

See what happens and check that `r` *and* `r5` *are again defined.*

c) You can also save variable definitions, procedures, or the code of complete worksheets.

Try the latter with a simple worksheet, using

File → Export As... → Maple Input (.mpl)

You can again `read` the code from the file. But the main application of this version is to run a Maple program (which requires no user interaction) in batch mode (typically for longer jobs).

Try this on `lva.student` *with a simple worksheet requiring no interactive input. Generate the* `.mpl` *file and start your job using*

`$ maple` *filename*`.mpl`