Winfried Auzinger
Dirk Praetorius

Sommersemester 2015
27.–29.05.2015

Übungsaufgaben zur VU Computermathematik

Serie 9

Eine Sammlung verschiedener Problemstellungen, teilweise mit stofflichen Ergänzungen, entsprechend erläutert. Enthält auch einige Grafik-Spielereien.

Exercise 9.1. *An example from linear algebra. Error handling.*

For a given matrix $A \in \mathbb{R}^{m\times n}$ with $m > n$, the matrix

$$A^+ := (A^T A)^{-1} A^T \in \mathbb{R}^{n\times m}$$

is called the pesudoinverse *of A. This is well-defined*[1] *if A has full rank m.*

a) Design a procedure `pinv(A::Matrix,v::Vector)`, which expects a matrix `A` and a vector `v` as its arguments and returns $A^+ v$. Use `LinearAlgebra[LinearSolve]`. Exit with an error message if the dimensions of `A` and `v` are not compatible. Test with numerical (`float`) data.

b) *The* `try: ... catch: ... 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.*[2]

Example:

```
try:
   b := 1/a;
   c := 1/d;
catch:
   error "Division by zero: a=0 or d=0!"
end try:
```

- Extend the procedure `pinv` by including a `try: ...` construct which issues your own error message (instead of the error message generated by `LinearSolve`) if system solving fails.

c) Modify `pinv` in an appropriate way such that, optionally, the matrix A^+ is returned.

Exercise 9.2. *Analyzing and visualizing a curve in 2D. (Use the* `plots` *package.)*

Consider the curve implicitly defined by the equation

$$\left(x^2 + y^2\right)^3 = 4\, x^2\, y^2$$

a) Use `plots[implicitplot]` to draw the curve. Produce a nice-looking plot by utilizing the options `thickness` and `numpoints`.

b) For an explicit parametric representation of this curve we use polar coordinates. Substitute $x = r\cos\varphi$, $y = r\sin\varphi$, and solve the given equation for r using `solve`. This gives $r = r(\varphi)$ as a function of φ.

Warning: In the natural polar representation, r also becomes negative. This can be seen from the animation **c)**.

Check the help page `?plot/parametric` and plot the curve using the polar representation $r = r(\varphi)$. Do the same using `plots[polarplot]`.

---

[1] $x = A^+ v$ is the solution of the so-called *Gaußian normal equations* $(A^T A)\, x = A^T v$. For this value x, the residual $\|A\, x - v\|_2$ becomes minimal. For $\operatorname{rank}(A) < m$ this solution x is not unique. In this case the pseudinverse A^+ is defined in another way.

[2] 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.

c) Use `plot` again to produce a sequence of plot structures `p[i]`, $i = 1, 2, \ldots$, representing sections of the curve (always beginning with $\varphi = 0$ and increasing the upper limit for φ from step to step). Then, use `display` with the option `insequence=true` to produce an animated plot (a small video).

Use the plot context menu on top to start the animation and to control it (in particular, the speed of the animation).

d) Compute the arclength of the curve according to the formula

$$\int_{\varphi=0}^{?} \sqrt{r(\varphi)^2 + r'(\varphi)^2}\, d\varphi\,.$$

This integral is not elementary; use `evalf(..)` or `int(...,numeric)` to evaluate it numerically. (The correct value is ≈ 9.6884.)

e) Compute the area of the interior of the curve by integrating its polar representation over one of the foils, e.g., the first one,

$$\int_{\varphi=0}^{?} \int_{\rho=0}^{|r(\varphi)|} \rho\, d\rho\, d\varphi\,.$$

This integral is elementary. (The correct value for the complete area is $\pi/2$.)

In **d)** and **e)**, check what the correct upper limit of the integral (?) is.

Exercise 9.3. *Some more 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. Consult `?animate` (look at the examples). Use animate to visualize the behavior of the functions $\left(1 + \frac{x}{n}\right)^n$ for $x \in [0, 5]$ and $n = 1, 2, \ldots$ (n is the parameter for the animation). Also, use an example of your own choice.

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` (see **2 c)**). 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 $\displaystyle\int_0^T f(t)\, dt$ with increasing T.

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

Exercise 9.4. *Devil's staircase.*

Consider the sequence of functions $D_n\colon [0,1] \to [0,1]$, defined by [3]

$$D_0(x) = x, \qquad \text{and} \qquad D_n(x) = \begin{cases} \frac{1}{2} D_{n-1}(3\,x), & 0 \le x < \frac{1}{3}, \\ \frac{1}{2}, & \frac{1}{3} \le x \le \frac{2}{3}, \\ \frac{1}{2}\big(1 + D_{n-1}(3\,x - 2)\big), & \frac{2}{3} < x \le 1, \end{cases} \qquad n \ge 1$$

a) Implement this sequence of functions in form of a procedure.

b) Play with `plot`, increase n, and zoom in (i.e., plot over small subintervals of $[0, 1]$).

c) Design a procedure `Devil(x)` which exactly evaluates the pointwise limit $\lim\limits_{n\to\infty} D_n(x)$ for rational `x` $\in [0,1] \cap \mathbb{Q}$.

Exercise 9.5. *Formatted I/O.*

a) Design a procedure which compresses a given matrix of `datatype=hfloat` using `CompressedSparseForm` and writes the compressed data to a text file. Choose an appropriate format.

b) Design a procedure for the reverse operation.

---

[3] These functions are continuous and monotonously increasing on $[0, 1]$. It can be shown that the pointwise limit $\mathrm{Devil}(x) := \lim\limits_{n\to\infty} D_n(x)$ is also a continuous and monotonously increasing.

For rational x, the sequence $\big(D_n(x)\big)$ becomes constant for sufficiently large n.

Exercise 9.6. *The package* `geometry`.

Consult the help page of the package `geometry`. In particular, check how to generate points, triangles, squares, etc., how to transform (translate, rotate, ...) them and how to draw them using `draw`.

Example: Check what happens here:

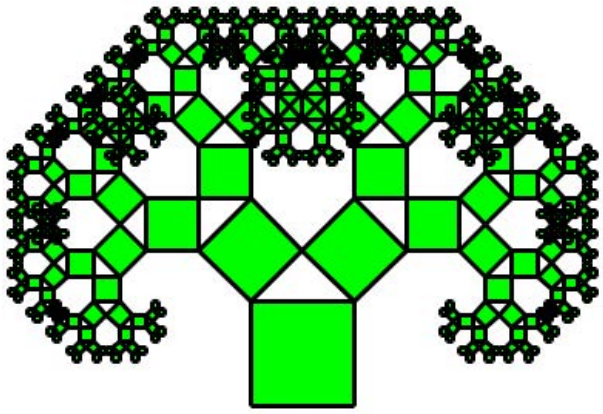
```
with(geometry):
with(plots):
unprotect(D); # unprotect the name D such it can be used to name a corner of the square
point(A,0,0), point(B,0,1), point(C,1,1), point(D,1,0);
square(S,[A,B,C,D]); # constructor generating the object S (a square)
p[1] := draw(S,color=black,axes=none,thickness=3);
p[2] := draw(S,color=green,axes=none,filled=true);
display(p[1],p[2]);
```

Typical syntax: The first argument is an output parameter.

Warning: Do not use floats for specifying coordinates; rounding errors are in conflict with internal operations of the `geometry` package and hinder exact identification of the type of an object. However, you may use algebraic numbers like $\sqrt{2}$ or multiples of π.

Warning: When performing transformations, e.g. `rotation(`*new_object,old_object,...*`)`, do not try to 'overwrite' data structures, i.e., do not use the same variable for *old_object* and *new_object*. This does not seem to work.

Exercise 9.7. $(*)$ *Drawing a fractal object using recursion, using the package* `geometry`.[4]



Design a recursive procedure `squares(level,s)` which generates a plot of a 'fractal' object (see figure). The arguments:

- `level` specifies the desired number of recursive levels.
- `s` is a `geometry` object of type `square` which is used to start the recursion.

The recursion works in the following way.

```
squares(level,s)
```

does the following job:

(i) Draw the square specified by `s`.

(ii) If `level>1`: Perform two recursive calls of `squares` with `level-1` which correspond to two smaller squares, rotated and sitting on top of the current square. For convenience, use the transformations (`rotation` etc.) from the `geometry` package – this saves much hand work.

Hint: Use global variables `p` and `counter`. Before calling `squares`, these are initialized: `p:='p'`, and `counter:=0`. Within `squares` the value `counter` is increased by 1, the plot of the current square is generated using `geometry[draw]`, and the resulting plot structure is saved in `p[counter]`. After calling `squares` you can render these plots using `plots[display]`.

---

[4] Zählt für 2 Aufgaben; beinhaltet ein bisschen Aufwand für selbständiges Erarbeiten von Teilen des `geometry` packages.
Die Grafik wurde mit dem Aufruf `squares(9,`*`Einheitsquadrat`*`)` generiert.