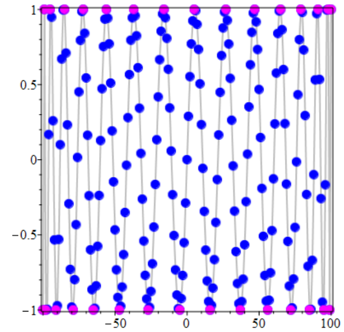
Winfried Auzinger
Dirk Praetorius

Sommersemester 2018
6. Juni 2018

Übungsaufgaben zur VU Computermathematik

Serie 9



A mixed collection of exercises.

Exercise 9.1: ***Parametric plots.***

a) *Consult the help pages and find out how to plot planar or spatial curves given in parametric representation,*

$$\{(x(t), y(t)),\ t \in [a,b]\} \qquad \text{or} \qquad \{(x(t), y(t), z(t)),\ t \in [a,b]\}\,.$$

Try some simple examples.

See `? plot/parametric`, `? plots[spacecurve]`; see also `? plots[tubeplot]`.

b) *Solve a system two of linear differential equations, e.g.,*

```
dsolve([D(x)(t)=-2*y(t),D(y)(t)=x(t)/2,x(0)=1,y(0)=1],[x(t),y(t)])
```

and plot the solution $(x(t), y(t))$ over an interval $t \in [0,T]$.

c) *Same as* **b)**, *for a system of three linear differential equations for $x(t)$, $y(t)$, $z(t)$.*

d) *Consult* `? plot3d` *and present some nice example.*

Exercise 9.2: ***Animated plots.***

a) The integral

$$F(t) = \int_0^t f(s)\,ds$$

is a function of t. For $f(t) \geq 0$, it represents the area between the t-axis and the graph of f.

Choose $f \geq 0$. Use `plot`, *and* `display` *with option* `insequence=true`, *to produce an animated visualization of $F(t)$ with varying t. Use the* `plot` *option* `filled=true` *to colorize the area under the graph of f.*

You may also use `? animate`.

b) *Present your own nice example (not simply one from the help page) using* `? animatecurve`.

Exercise 9.3: ***A special point plot.***

Let a list of pairs (x_i, y_i), $i = 1 \ldots n$ (with float values) be given, where each pair is represented by a list of length 2. Plotting the corresponding points $(x_i, y_i) \in \mathbb{R}^2$ using `pointplot` is straightforward. However, assume that we wish to highlight special values in such a plot.

Produce a pointplot where the points (x_i, y_i) with

$$|y_i - 1| \leq \varepsilon \quad \textit{or} \quad |y_i + 1| \leq \varepsilon$$

are displayed in a color different from the color used for the other points.

You may, e.g., choose $\varepsilon = 10^{-2}$, and use blue color, and red color for highlighting. See the example in the plot above.

Exercise 9.4: ***Numerical approximation with error estimation.***

Assume that a function $f(t)$ satisfies $f(0) = f'(0) = \ldots = f^{(n-1)}(0) = 0$ for some $n \in \mathbb{N}$. Then, by Taylor's theorem,

$$f(t) = \frac{t^n}{n!}\, f^{(n)}(0) + O(|t|^{n+1}) \quad \text{for} \quad t \to 0\,.$$

For such a function f, we can approximate the integral

$$\int_0^t f(s)\, ds$$

over a (small) interval $[0,t]$ by

$$\int_0^t \frac{t^n}{n!}\, f^{(n)}(0)\, dt = \frac{t^{n+1}}{(n+1)!}\, f^{(n)}(0) \approx \frac{t}{n+1}\, f(t)\,, \tag{Q}$$

which involves only a single evaluation of f.

We now consider a practical application of this integral approximation: Assume that we are approximating the function[24] e^t by a rational function $r(t)$. We choose a so-called (3,3) Padé approximation,

$r(t) =$ `numapprox[pade](exp(t),t,[3,3])`.

a) *Verify using* `taylor` *that* $r(t) = e^t + O(|t|^7)$ *for* $t \to 0$.

b) The function e^t is the solution of the differential equation $y'(t) - y(t) = 0,\ y(0) = 1$.

Compute the residual of $r(t)$ *with respect to this differential equation, i.e., the rational function*

$$\delta(t) := r'(t) - r(t)\,.$$

Verify using `taylor` *that* $\delta(0) = \delta'(0) = \ldots = \delta^{(5)}(0) = 0$ *and* $\delta^{(6)}(0) \neq 0$, *i.e.,* $\delta(t) = O(|t|^6)$ *for* $t \to 0$.

c) Due to $e^0 = r(0) = 1$ and by definition of $\delta(t)$, the approximation error $\varepsilon(t) := r(t) - e^t$ satisfies the differential equation

$$\varepsilon'(t) = \varepsilon(t) + \delta(t)\,, \quad \varepsilon(0) = 0\,,$$

which implies

$$\varepsilon(t) = \int_0^t \underbrace{e^{(t-s)}\,\delta(s)}_{=:\, f(s;t)}\, ds\,.$$

Verify that the integrand $f(s;t)$ *(considered as a function of* s *for fixed* t*) satisfies* $f(0;t) = f'(0;t) = \ldots = f^{(5)}(0;t) = 0$ *and* $f^{(6)}(0;t) \neq 0$.

d) In order to obtain an easily computable estimate for the error $\varepsilon(t) = r(t) - e^t$ (control of accuracy!), we approximate its integral representation by the quadrature formula (Q), i.e., we compute

$$\tilde{\varepsilon}(t) := \frac{t}{7}\, f(t;t) = \frac{t}{7}\, \delta(t) \approx \varepsilon(t)\,.$$

Verify using `taylor` *that*

$$\tilde{\varepsilon}(t) - \varepsilon(t) = O(|t|^8) \quad \text{for} \quad t \to 0\,.$$

This shows that the error estimate is very precise for sufficiently small t, with a relative deviation of size $O(t)$. *Visualize this by plotting the true error* $\varepsilon(t) = r(t) - e^t$ *and its estimate* $\tilde{\varepsilon}(t)$ *for* $t = 0 \ldots 1$. *Use also* `plots[logplot]` *for a better visualization of the asymptotic behavior for* $t \to 0$.

---

[24] This is a basic example. In a practical setting we would e.g. consider an approximation of the matrix exponential e^{tA}, which may be difficult to compute exactly if the dimension of A is large.

Exercise 9.5: `map`, `map2`; `heap`.

a) *Explain, by be means of a small example worksheet, the use of* `map` *and* `map2`*.*

b) A heap is an data structure (usually implemented using a tree structure) for dynamically storing objects from a (in practice: very large) ordered set, in such a way that efficient direct access to the maximal element is provided (for seeking or removing it). New objects can be inserted into an existing heap.

Check the help page of the small package `? heap`*. Look at the example provided on the help page. Generate the empty heap with ordering function* `f:=(a,b)->evalb(a<b)` *for storing integer values. The heap will now internally store the elements in sorted order. Then, insert a number of integers in random order and extract the maximal element.*

Remark: The dynamical data structures `queue` and `stack` are also available, for FIFO (First In, First Out) access and LIFO (Last In, First Out) access, respectively. Usage is similar as for heaps.

Exercise 9.6: ***File handling.***

a) *Take one of your worksheets which requires no interactive input and export it to a Maple language (text) file (file type* `.mpl`*). Then execute it using* `maple` *(or* `cmaple`*) and redirect the output to another text file.*

Remark: This is useful if longer jobs have to run in batch mode. Depending on your installation, the command line version of Maple is called `maple` or `cmaple`. If this is not in your execution path, check its location.

b) *Write a short procedure and use* `save` *to store its source text on a* `.mpl` *file. Change this file with a text editor and read it back to your worksheet using* `read`*. Check what happens if your code contains a syntax error.*

c) Assume that the coefficients of an $m \times n$-matrix consisting of columns of a given (a priori known) length m stored on a text file. Each line contains one column of the matrix (values separated by spaces).

Design a procedure which expects the name of the text file as its argument and which returns the matrix in form of an object of type `Matrix`*.*

Hint: Use `? readline` and `? sscanf`

Exercise 9.7: ***Try to catch.***

Similarly as in other languages (e.g., C or MATLAB), `try .. catch` is a convenient construct to supervise the execution of parts of a code where successful execution is not a priori guaranteed and some error may occur. Typically this is used within procedures.

```
try
  ...
  ...  # do something; if it is O.K. then it is O.K.
  ...
catch:
  # specify what has to be done if try has failed, e.g.
  error("oops, this does not work"):
  # or some alternative part of code to be executed:
  ...
  ...
end try:
```

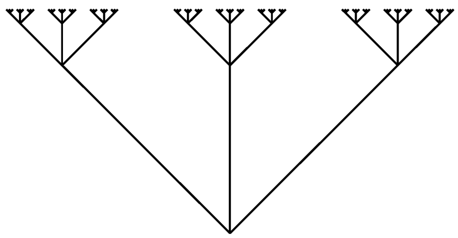
We realize a very simple example:

Use the `try .. catch` *construct twice (in a nested way) within a procedure which tries to invert a given matrix* `A`*. If straightforward inversion fails, your procedure tries to compute the so-called pseudo-inverse* $(A^T A)^{-1} A^T$*. If this also fails, then your procedure exits with an error message. Find (numerical) examples for all three cases.*

This is connected with a question about basic linear algebra: What type of failure occurs for what type of matrix? Do you know what the pseudo-inverse represents? [25]

[25] You can nest this further and return a (generalized) pseudo-inverse which is well-defined for any matrix A. But this is out of scope here.

Exercise 9.8: ***Drawing a fractal tree.***



Design a recursive procedure `tree`(*level,pos,n,len,scale*) *which generates a plot of a 'fractal' tree.*

The arguments:

- *level* specifies the desired number of recursive levels.
- *pos* is a list of length 2 containing the cartesian coordinates of the root of the current tree.
- The current tree consists of n branches emanating from the root with randomly chosen inclinations (angles between $-45°$ and $+45°$), and *len* specifies the current (common) length of these branches.
- *scale* is a fixed factor with which *len* is multiplied when proceeding to the next recursive level.

The recursion works in the following way.

`tree`(*level,pos,n,len,scale*)

does the following job:

(i) Generate the n branches of the tree specified by the arguments *pos*, n, and *len*.

(ii) If $level > 1$: Perform n recursive calls of tree with *level-1*, n new values for *pos* which correspond to the endpoints of the branches of the current tree, and with *len* replaced by $scale * len$.

The figure displayed above was generated by calling `tree` with $n = 3$, $scale = 1/4$ and non-randomly chosen (fixed) angles.

Hint: See `?rand`. Use global variables `p` and `counter`. Before calling `tree`, these are initialized: `p:='p'`, and `counter:=0`. When `tree` builds up a branch, the value `counter` of is increased by 1, the plot of the branch is generated using, e.g., `plots[pointplot]`, and the resulting plot structure is saved in `p[counter]`. After activating `tree` you can render these plots using `plots[display]`. Playing with the parameters you can generate different funny trees.