Winfried Auzinger
Kevin Sturm

Sommersemester 2019
10.04.2019

Übungsaufgaben zur VU Computermathematik

Serie 5

A mixed collection of exercises on different topics.

Exercise 5.1: ***A stability investigation.***

Consider the trivial differential equation $y'(t) = 0$ with given initial value $y(0) = 0$, such that the solution is $y(t) \equiv 0$. This serves as a simple model problem for the following stability investigation.

Mr. Rembremerdeng proposes to approximate the solution $y(t)$ at integer values $t = n \in \mathbb{N}$ by the three-step recursion

$$\big(y(n) \approx \big) \;\; y_n := \frac{18}{11}\, y_{n-1} - \frac{9}{11}\, y_{n-2} + \frac{2}{11}\, y_{n-3}\,, \quad n = 3, 4, \ldots \tag{1}$$

Here we need three initial values y_0, y_1, y_2. For $y_0 = y_1 = y_2 = 0$ the solution is $y_n \equiv 0$ (it is exact). Now assume that these zero initial values are perturbed a little bit, e.g., by rounding errors. We ask: What is the effect of such a perturbation?

a) *First, perform a numerical experiment to investigate the behavior of the sequence $\big(y_n\big)$ in floating point arithmetic for increasing values of n.* (For what range of values of n you observe a significant effect depends on the size of the initial perturbations.)

b) Now we try to understand theoretically what we have observed in **a)**.

Determine λ_1, λ_2, and λ_3 such that the general solution of (1) *(for given initial values y_0, y_1, y_2) can be represented in the form*

$$y_n = c_1\, \lambda_1^n + c_2\, \lambda_2^n + c_3\, \lambda_3^n$$

with constants c_1, c_2, c_3 (which will depend on y_0, y_1, and y_2).

Hint: You can guess one of the solutions (λ_1) and reduce the problem to a quadratic equation. Or simply use `solve`.

c) Is such a solution asymptotically stable, i.e., does it remain uniformly bounded for $n \to \infty$?

You can answer this question by just inspecting the λ_k; however, λ_2, λ_3 form a complex conjugate pair. If you are not sure, plot the absolute values of the λ_k^n for increasing n.

Exercise 5.2: ***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).

Exercise 5.3: ***Exploring the behavior of a sequence via experiment.***

We consider sequences (x_n) defined in a recursive way by

$$x_n := f(x_{n-1}), \quad n = 1, 2, 3, \ldots, \quad \text{with} \quad f(x) = \frac{1}{2}\left(x - \frac{1}{x}\right)$$

starting from a given initial value x_0. We observe:

- For $x_0 = 0$ we immediately end up with $x_1 = \infty$.
- For $x_0 = \pm 1$ we have $x_1 = 0$ and $x_2 = \infty$.
- For all other $x_0 \in \mathbb{Q}$, the sequence (x_n) is well-defined for all n. (*Why? This is simple to prove. Note that for $x_0 \in \mathbb{Q}$, all x_n are rational numbers.*)
- For complex $x_0 \in \mathbb{C}$, $x_0 \notin \mathbb{R}$, the sequence (x_n) is well-defined, with $x_n \notin \mathbb{R}$ for all n.
- For $x_0 \in \mathbb{R}$, the (real) sequence (x_n) cannot be convergent, since the only possible limits are i and $-$i. (*Can you explain this?*)

a) *Design a procedure which, for given $n \in \mathbb{N}$, produces a plot of the points (n, x_n) for given x_0.*

Hint: For `pointplot`, a recommended set of options is

`style=point`, `axes=boxed`, `symbolsize=20`, and `symbol=solidcircle`

b) Conjecture: For all $x_0 \in \mathbb{C}$ with $\operatorname{Im} x_0 > (<)\ 0$, the iteration converges to $\pm$i. We explore this experimentally:

Design a procedure which expects $x_0 \in \mathbb{C}$, $\varepsilon > 0$ and $n_{max} \in \mathbb{N}$ as its arguments and which returns the minimal value $n \in \mathbb{N}$ such that $|x_n - (\pm \mathrm{i})| \le \varepsilon$. If no such $n \le n_{max}$ is detected, use `error` *to issue an error message including the value of $x_{n_{max}}$. Use* `evalf`*.*

Play with your procedure, in particular with x_0 very close to 0.

Exercise 5.4: ***Simulating the movement of a pendulum.***

We consider the movement of a pendulum, described by its angle of deflection $\varphi = \varphi(t)$ as a function of time t. The governing diffferential equation is

$$\ddot{\varphi}(t) = -\sin(\varphi(t))$$

where $\ddot{\varphi}$ is the second derivative of φ w.r.t. t. Together with initial conditions for φ and $\dot{\varphi}$, e.g.,

$$\varphi(0) = 1, \quad \dot{\varphi}(0) = 1$$

the problem has a unique solution $\varphi(t)$ for all t, but the solution cannot be represented in an exact, analytic form. Therefore we investigate some numerical methods. First of all, we introduce the angular velocity $\psi(t) := \dot{\varphi}(t)$ as a separate variable and consider the equivalent system

$$\begin{pmatrix} \dot{\varphi}(t) \\ \dot{\psi}(t) \end{pmatrix} = \begin{pmatrix} \psi(t) \\ -\sin(\varphi(t)) \end{pmatrix} \quad \text{with initial conditions} \quad \begin{pmatrix} \varphi(0) \\ \psi(0) \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

a) We choose a timestep h and use the simplest numerical scheme in order to compute approximations (φ_n, ψ_n) to the solution $(\varphi(t_n), \psi(t_n))$ at the times $t_n := n\,h$, $n = 0, 1, 2, \ldots$. To this end we replace the derivative $\dot{\varphi}(t_n)$ by the forward difference quotient, i.e.,

$$\dot{\varphi}(t_n) \approx \frac{\varphi_{n+1} - \varphi_n}{h}$$

and analogously for ψ. This leads to the recursion

$$\begin{pmatrix} \varphi_{n+1} \\ \psi_{n+1} \end{pmatrix} = \begin{pmatrix} \varphi_n \\ \psi_n \end{pmatrix} + h \begin{pmatrix} \psi_n \\ -\sin(\varphi_n) \end{pmatrix}, \quad n = 0, 1, 2, \ldots, \quad \text{starting from} \quad \begin{pmatrix} \varphi_0 \\ \psi_0 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

Implement this method by a simple loop, using `evalf`*, to generate a list of vectors containing the solution values (φ_n, ψ_n) at the times t_n, $n = 0, 1, 2, \ldots$. Choose the timestep $h = 0.1$ and produce a pointplot of the (φ_n, ψ_n), $n = 0, 1, 2, \ldots$, for n up to 200 (i.e., $t = 20$). The solution should behave periodic. What do you observe?*

b) For larger t-intervals, the method from **a)** produces a qualitatively incorrect approximation. Here is a simple remedy: We use the modified recursion

$$\begin{pmatrix} \varphi_{n+1} \\ \psi_{n+1} \end{pmatrix} = \begin{pmatrix} \varphi_n \\ \psi_n \end{pmatrix} + h \begin{pmatrix} \psi_n \\ -\sin(\varphi_{n+1}) \end{pmatrix}$$

Repeat the experiment from **a)** *using the modified recursion. What do you observe?*

c) *In addition, apply* `dsolve` *with option* `numeric` *(without further special settings), and use* `plots[odeplot]`*:*

```
sol:=dsolve([D(u)(t)=v(t),D(v)(t)=-sin(u(t)),u(0)=1,v(0)=1],[u(t),v(t)],numeric)
plots[odeplot](sol,[u(t),v(t)],t=0..200,axes=boxed,thickness=2)
```

What do you observe? Also extend the range from $t = 20$ *to* $t = 300$ *and repeat* **a)**, **b)**, *and* **c)**.

Remark: `dsolve/numeric` delivers a procedure, and calls to this procedure can be use to evaluate the numerical approximation at particular particular times t, or it can be passed to `? plots[odeplot]`.

Exercise 5.5: *Matrix representation of a linear mapping.*

Design a procedure `GenerateMatrix(f::procedure)`

which expects a function `f`: $\mathbb{R}^n \to \mathbb{R}^m$ *(in form of a procedure) as its argument and which returns the corresponding coefficient matrix* $A \in \mathbb{R}^{m \times n}$. *Test with an example of your choice.*

(Assume a priori that `f` indeed is linear and maps `Vector`s into `Vector`s.)

Exercise 5.6: *Unapply.*

Assume that you generate a complicated symbolic formula $f(x)$, and later on you need to evaluate it numerically for many different numerical values of x within a loop. Here it is very inefficient to repeat the symbolic computation again and again within the loop.[1]

To this end, one can use `? unapply`. *Check this and prepare a tutorial on the topic. Choose your own example, implement and test it, and demonstrate.*

Exercise 5.7: *Try / catch.*

Similarly as in other modern programming languages (e.g., C, PYTHON, MATLAB, JULIA, ...), the `try .. catch` construct is convenient for supervising sections of a code where successful execution is not a priori guaranteed but some error (which nay be difficult to predict) may occur.

This is typically 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 for some reason, e.g.
  error("oops, this does not work"):
  # or some alternative part of code to be executed:
  ...
end try:
```

Choose your own example, implement it, and test it.

Exercise 5.8: *Implicit plots.*

Curves in the (x,y)- plane are often specified in an implicit way, i.e., via a functional relation $f(x,y) = 0$. Finding an explicit representation of the curve may be not so easy (or impossible), but `? plots[implicitplot]` tries to do a good job using a numerical pathfollowing algorithm.

Consider the curve defined by

$$f(x,y) = (x^2+y^2)^3 - 4\,x^2\,y^2 = 0,$$

and produce a nice plot.

---

[1] This topic has been discussed in the lecture: The point is that symbolic computation usually performs much slower than numerical computation.