Winfried Auzinger
Dirk Praetorius

Sommersemester 2014
4.–6. Juni 2014

Übungsaufgaben zur VU Computermathematik

Serie 9

Eine Sammlung verschiedener Problemstellungen. Die Angaben zu manchen der Aufgaben sind relativ ausführlich geraten, zwecks Erläuterung des anwendungsorientierten Hintergrundes. Es sieht daher nach mehr Arbeit aus als es wirklich ist.

Exercise 9.1. Consider the trivial differential equation $y'(t) = 0$ with 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.) We approximate the solution by the three-step recursion [1]

$$y_n := \frac{18}{11}\, y_{n-1} - \frac{9}{11}\, y_{n-2} + \frac{2}{11}\, y_{n-3}\,, \quad n = 3, 4, \ldots \tag{1}$$

We need three initial values y_0, y_1, y_2. For $y_0 = y_1 = y_2 = 0$ the solution is $y_n \equiv 0$. Now assume that the initial values are perturbed to be $\neq 0$, e.g., by rounding errors. We ask: What is the effect of such a perturbation?

a) Exercise **7.5** (two-step recursion) readily generalizes to the case of the present three-step recursion. Determine λ, λ_2 and λ_3 such that the general solution of (1) (for arbitrary initial values) is given by

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

with arbitrary constants c_1, c_2, c_3.

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

b) Is such a solution *stable*, i.e., does it remain uniformly bounded for $n \to \infty$? (You can answer this question by just inspecting the the λ_i; however, λ_2, λ_3 form a complex conjugate pair. If you are not sure, plot the absolute values of the λ_i^n for increasing n.)

Exercise 9.2. Use 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).

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 $f(t)$ 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.

---

[1] This recursion emerges if one approximates y' in a similar way as in 7.7. A related recursion can be used to approximate more general differential equations.

d) Also check `animate3d` and show some nice example. You may also try to combine several calls of `plot3d` in combination with `display3d(...,insequence=true)`.

Exercise 9.3. Consider the trefoil (*Kleeblatt*) curve implicitly defined by the equation

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

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 the trefoil curve we use polar coordinates. Substitute $x = r\cos\varphi,\ y = r\sin\varphi$, solve the trefoil equation for r using `solve`, and simplify the result. This gives $r = r(\varphi)$ as a function of φ, and $x(\varphi) = r(\varphi)\cos\varphi$, $y(\varphi) = r(\varphi)\,\sin\varphi$.

c) Use `plots[animatecurve]` to draw the curve once more. (Since we are now using the parametric representation, you have to use `animatecurve` in a way analogous as described in `? plot/parametric`.)

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

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

Be careful: What is the correct upper limit for the integral? (Look at your plot from **b)**.)

The integral is not elementary; use `evalf` to evaluate it numerically. (The correct value is ≈ 6.682.)

e) Compute the area of the trefoil 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 (but requires some work doing it by hand). (The correct value for the area of the trefoil is $\pi/4$.)

Exercise 9.4.

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

b) Check the help page `? index` and look for `packages`. Here you see a complete list of available packages. As an example, we look at the small package `heap`. A heap is an data structure for dynamically storing objects from a (in practice: large) ordered set, in such a way that efficient direct access to the maximal element is provided. Any object my be inserted into an existing heap. Look at the example provided on the help page. Produce a small worksheet, use `with(heap)`; and generate a heap for storing integers. Generate the empty heap with ordering function `f:=(a,b)->evalb(a<b)`. 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: In addition, the dynamical data structures `queue` and `stack` are available for FIFO (First In, First Out) access and LIFO (Last In, First Out) access, respectively. Usage is similar as for heaps.

Exercise 9.5. Assume you want to do symbolic computations with abstract objects for which multiplication is not commutative. (Think of the algebra of real or complex $n \times n$-matrices.) Noncommutative multiplication is supported in the package `Physics`.

By

```
with(Physics):
Setup(noncommutativeprefix={A,B,C}):
Setup(mathematicalnotation=true):
Setup(noncommutativecolor=green):
```

you activate the package, declare the variables `A,B,C` as non-commutative, and activate green color for the display of non-commutative objects. The `Physics` command [2] `Commutator(A,B)` computes the so-called `commutator` $[A, B]$ of A and B,

$$[A, B] := A\,B - B\,A\,.$$

a) Use `Commutator` and [3] `Simplify` to verify the fundamental identities

$$[A\,B, C] = A\,[B, C] + [A, C]\,B$$

and

$$[A, [B, C]] + [B, [C, A]] + [C, [A, B]] = 0 \qquad \text{(Jacobi identity)}.$$

b) For the definition of the matrix exponential [4] e^A, see **8.8**. In applications, something like $e^{t(A+B)}$ (with $t \in \mathbb{R}$ and two matrices A, B) may be hard to compute numerically, while e^{tA} and e^{tB} are simpler to compute. We consider the *Lie-Trotter approximation*

$$L(t) := e^{tA}\,e^{tB} \approx e^{t(A+B)}\,.$$

(Think of t as a small time step.) We now study the structure of the error of this approximation and its dependence on t for $t \to 0$. Consider the quadratic Taylor polynomials [5]

$$T_A(t) = I + t\,A + \frac{t^2}{2}\,A^2\,, \quad \text{and} \quad T_B(t),\, T_{A+B}(t)\,,$$

and consider $T_A(t)\,T_B(t) - T_{A+B}(t)$. Use `Simplify` to verify that that the leading term (with lowest power of t) is $O(t^2)$ and identify this leading term as commutator expression.

(Remark: If A and B commute, i.e., $[A, B] = 0$, then the Lie-Trotter approximation is exact.)

c) The *Strang approximation*

$$S(t) := e^{t/2\,A}\,e^{tB}\,e^{t/2\,A} \approx e^{t(A+B)}$$

is is of a better quality. Proceed analogously as in **b)**, considering the Taylor polynomials $T_{A/2}(t)$, $T_B(t)$ and $T_{A+B}(t)$ of degree 3. Use `Simplify` to verify that that the leading term of $T_{A/2}(t)\,T_B(t)\,T_{A/2}(t) - T_{A+B}(t)$ is $O(t^3)$. Verify that it admits the representation

$$t^3\left(\tfrac{1}{24}\,[[A, B], A] + \tfrac{1}{12}\,[[A, B], B]\right).$$

---

[2] You may define an abbreviation for this, e.g., `alias(K=Commutator)`.

[3] `Simplify` is the simplification command from the `Physics` package.

[4] Remark: For some given vector u_0, the vector-valued function $u(t) = e^{tA}\,u_0$ is the solution of the vector-valued differential equation $u'(t) = A\,u(t)$, $u(0) = u_0$.

[5] Do not use `taylor` – this is not correctly supported in `Physics`. (Bug?)

Exercise 9.6. Some advanced programming features:

- The `overload` command allows you to split the implementation of a command operating on different types of arguments into separate procedures. The basic syntax is `overload(P)`, where `P` is a list of procedures.

  A simple example: Assume you want to represent matrix left division $A\backslash B := A^{-1} \cdot B$ and scalar division $a^{-1}\, b$ by a single function `backslash`. This works as follows:

```
backslash := overload([
             proc(A::Matrix,B::Matrix)
               option overload:
               return A^(-1).B
             end proc,
             proc(a::anything,b::anything)
               option overload:
               return a^(-1)*b
             end proc
             ]):
```

  The first branch only accepts arguments of type `Matrix`. The data type `anything` means 'any possible type', i.e., the second procedure acts as a default.

  Such a construction is often more useful than embedding various `if`-constructs into a single procedure.

- The `try ... catch ... end try` construct allows you to 'protect' parts of your code, with a controlled error handling by the `catch`-branch if the `try`-branch fails. A simple example:

```
try:
   M := Matrix(m,m):
   N := Matrix(n,n):
catch:
   error "One of the matrix dimensions has not been correctly specified."
end try:
```

  This is often more useful than trying to avoid in advance, by various `if`-constructs, that such an error occurs.

a) Verify the above simple example for the use of `overload`, and extend it by a `try ... catch ... end try` construct which, for the case that when dividing by `A` or `a`, respectively, an error occurs, the `catch`-branch forces a return of `FAIL`.

b) Extend `backslash` in such a way that for a square matrix `A` and a vector `b` the solution x of the linear system $A\,x = b$ is returned. Use `LinearAlgebra[LinearSolve]`.

Exercise 9.7. We implement a method for computing the matrix exponential e^A based on the spectrum of A, which is practically applicable if the dimension of A is small.

a) For pairwise distinct nodes $x_1, \ldots, x_n$ and values $y_1, \ldots, y_n$ there exists a unique *interpolating polynomial* of degree $\leq n-1$ with $p(x_i) = y_i,\ i = 1 \ldots n$. $p(x)$ can be computed in different ways. We use the following method: Define the *Lagrange polynomials*

$$L_i(x) := \frac{(x-x_1)\ldots'(x-x_n)}{(x_i-x_1)\ldots'(x_i-x_n)}\,, \quad i = 1 \ldots n,$$

where $\ldots'$ means that the product does not include the index i. The $L_i(x)$ satisfy $L_i(x_j) = \delta_{i,j}$, which in turn implies the representation

$$p(x) = \sum_{i=1}^{n} y_i\, L_i(x)\,.$$

Design a procedure `interp(X,Y)` which expects two lists `X` and `Y` as its arguments and returns the function $p(x)$.

Hint: This is a floating point algorithm. Generate the formula for $p(x)$, use `simplify, collect(...,x)` and `evalf`, and return `unapply(...,x)`.

b) Let $A \in \mathbb{R}^{n\times n}$ be a diagonalizable matrix with n distinct eigenvalues $\lambda_i,\ i = 1 \ldots n$. Let $p(x) = c_0 + c_1\, x + \ldots + c_{n-1}\, x^{n-1}$ be the polynomial of degree $\leq n-1$ satisfying $p(\lambda_i) = e^{\lambda_i}\ \ i = 1 \ldots n$ (see **a**)). Then,

$$e^A = p(A) = c_0\, I + c_1\, A + \ldots c_{n-1}\, A^{n-1}\,.$$

Implement this in form of a procedure `mexp(A)`. Use `LinearAlgebra[Eigenvalues](evalf(A))` to compute the λ_i.

Test a 3×3 example and compare with the result provided by
`LinearAlgebra[MatrixExponential](evalf(A))`.

Hint: This is a floating point algorithm. Evaluate $p(A)$ using the Horner scheme (see **8.8**).

Exercise 9.8. 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,

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

For such a function f, we can approximate the integral

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

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

$$\int_0^t \frac{t^n}{n!}\, f^{(n)}(0)\, dt = \frac{t^{n+1}}{(n+1)!}\, f^{(n)}(0) \approx \frac{t}{n+1}\, f(t)\,, \tag{2}$$

which involves only a single evaluation of f.

We now consider a a practical application of this integral approximation: Assume that we are approximating the function [6] e^t by a rational function $r(t)$. We choose a Padé approximation (see **8.8**),

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

[6] This is a baby 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 A has a larger dimension.

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

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

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 (2), 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, 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 $t \to 0$.
